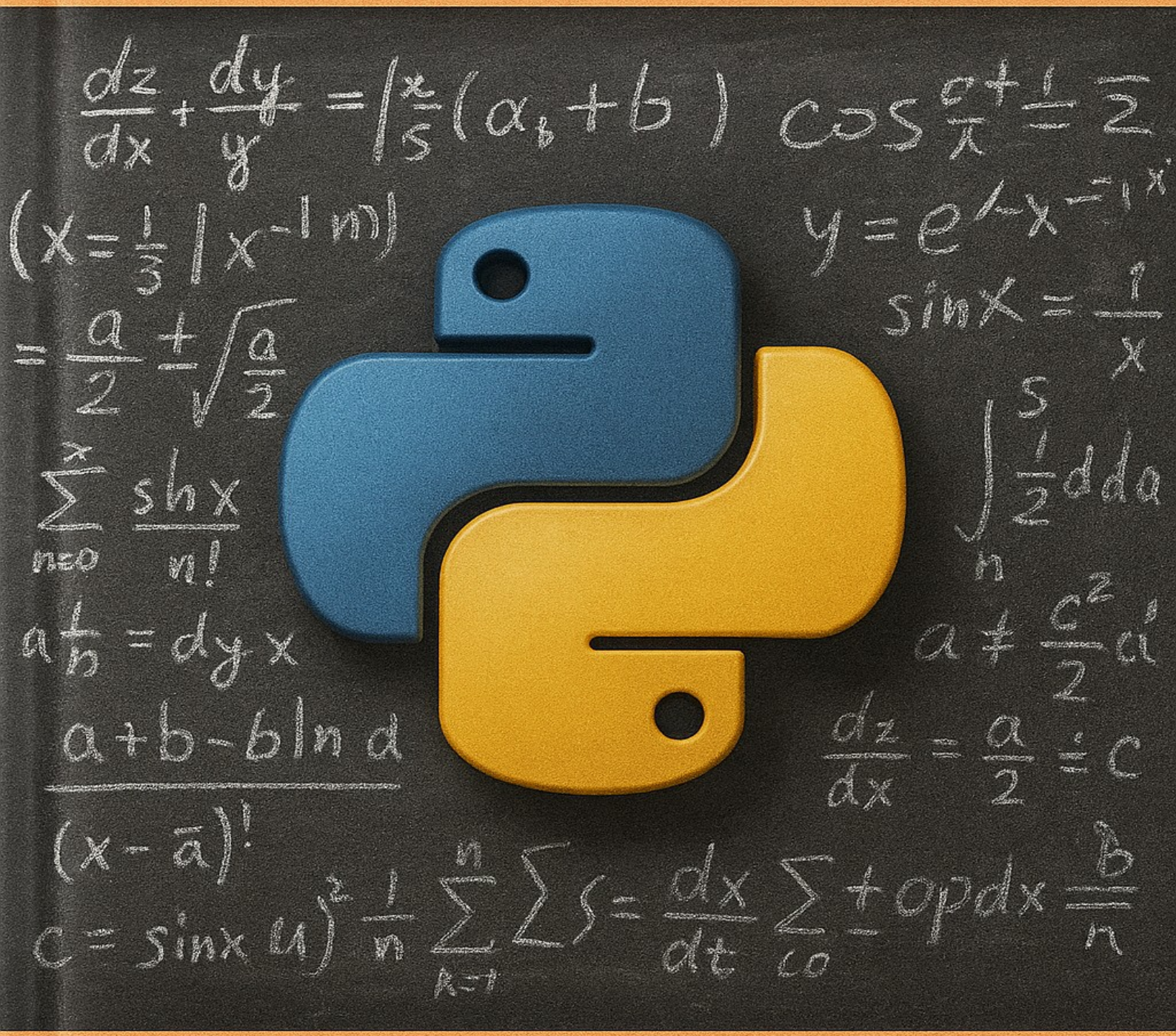
MATEMÁTICAS PARA PYTHON



Luis Minguillón Pascual

Matematicas para Python

Luis Minguillón Pascual

COPYRIGHT

La licencia de este libro eslectrónico es para uso personal. Por lo tanto no puedes revenderlo a otras personas.

Gracias por respetar los derechos de autor.

©Luis Minguillón Pascual
14/11/2025

Prefacio

La programación y las matemáticas siempre han estado unidas por un hilo invisible que impulsa el progreso tecnológico y científico. Este libro nace precisamente de esa conexión: es una guía diseñada para quienes desean aprender matemáticas a través de la programación y programación a través de las matemáticas. Aquí aprenderás a utilizar **Python**, un lenguaje sencillo, versátil y poderoso, para abordar conceptos matemáticos de forma práctica, visual y profunda.

El propósito de este libro es ayudarte a comprender las matemáticas no solo como teoría, sino como una herramienta viva que se convierte en código, algoritmos y soluciones concretas. Cada capítulo te introducirá en ideas matemáticas fundamentales y te mostrará cómo traducirlas a Python, permitiéndote experimentar, calcular, simular y resolver problemas reales con tus propias manos.

La importancia de las matemáticas es innegable: son el lenguaje con el que entendemos el mundo, modelamos fenómenos, optimizamos procesos y estructuramos el pensamiento lógico. Por su parte, la programación es la herramienta que nos permite convertir ese lenguaje en acción, automatización y creatividad aplicada. Python, con su sintaxis clara y su enorme ecosistema de bibliotecas, se ha convertido en el punto de encuentro ideal entre ambos mundos. Es hoy uno de los lenguajes más utilizados en ciencia, educación, inteligencia artificial y análisis de datos.

Este libro es, por tanto, una invitación: a explorar, a construir, a equivocarte y a descubrir. Que cada ejercicio y cada línea de código te acerquen a un entendimiento más profundo tanto de las matemáticas como del acto de programar. Siéntete libre de experimentar, modificar los ejemplos y llevar cada idea más allá. La mejor forma de aprender es creando.

Bienvenida a este viaje donde las matemáticas toman forma en Python y donde Python te abre una puerta nueva hacia la comprensión del universo matemático.

Luis Minguillón Pascual

ÍNDICE

1. LECCIÓN: SISTEMAS DE NUMERACIÓN

- Introducción a los sistemas numéricos: decimal (base 10), binario (base 2), octal (base 8) y hexadecimal (base 16).
- Conversión entre sistemas numéricos.
    - Uso de funciones integradas de Python: `bin()`, `oct()`, `hex()`.
    - Conversión inversa (a decimal) usando `int(x, base)`.
    - Formato de salida personalizada (`format()` o f-strings).
    - Conversión de decimal a otras bases manualmente (ejemplo: 13 a binario).
- Operaciones con números binarios.
- Funciones útiles para bases numéricas (`int()`, `format()`, f-string).
- Uso en la programación (datos binarios, colores HTML, valores ASCII).
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE OTROS SISTEMAS DE NUMERACIÓN.

2. LECCIÓN: NÚMEROS ROMANOS

- Introducción a los números romanos.
    - Símbolos principales (I=1, V=5, X=10, L=50, C=100, D=500, M=1000).
    - Reglas principales (suma, resta, repetición).
    - Ejemplos de números romanos comunes.
- Conversión de decimal a romano (método de resta del valor más alto posible).
- Conversión de romano a decimal (reglas de suma y resta por posición).
- Combinando ambas conversiones (función doble).
- Errores comunes y validación de número romano (uso de expresiones regulares `re`).
- Aplicaciones prácticas.
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE NÚMEROS ROMANOS.

3. LECCIÓN: RELACIONES DE IGUALDAD Y DESIGUALDAD

- Introducción (comparaciones booleanas True o False).
- Operadores relacionales principales (`==`, `!=`, `>`, `<`, `>=`, `<=`).
- Importancia de los valores booleanos en estructuras condicionales (`if`, `elif`, `else`).
- Comparación entre distintos tipos de datos (números, cadenas).
- Combinación de relaciones (operadores lógicos `and`, `or`, `not`).
- Operaciones encadenadas (ejemplo: `5 < x < 15`).
- Comparaciones de flotantes (manejo de imprecisión usando `round()` o tolerancia).
- Comparaciones de texto con mayúsculas y minúsculas.
- Comparaciones de objetos (listas, tuplas, cadenas).
- Operadores **"is" y "is not"** (comparan identidad de objeto, no valor).
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE RELACIONES DE IGUALDAD Y DESIGUALDAD.

4. LECCIÓN: OPERACIONES ARITMÉTICAS

- Introducción (operadores con números enteros o decimales).
- Operadores aritméticos: suma (`+`), resta (`-`), multiplicación (`*`), división (`/`), división entera (`//`), resto/módulo (%), potencia (`**`).
- La suma y la resta.
- La multiplicación (también para repetir cadenas).
- La división (devuelve flotante).
- La división entera.
- El módulo (para obtener el resto, útil para saber si un número es par).
- La potencia.
- Raíces cuadradas y otras raíces (usando potencia fraccionaria).
- Prioridad de operaciones (paréntesis, potencias, mult/div/módulo, suma/resta).
- Funciones matemáticas adicionales (módulo `math`: `sqrt`, `pow`, `floor`, `ceil`).
- Conversión de tipos numéricos (`int()`, `float()`) y redondeo (`round()`).
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE OPERACIONES ARITMÉTICAS.

5. LECCIÓN: OPERACIONES CON NÚMEROS ENTEROS

- Introducción (tipo de dato `int`, positivo, negativo o cero).
- Características de los números enteros (no tienen límite fijo, se usan en condicionales y bucles).
- Operadores aritméticos básicos (Suma, Resta, Multiplicación, División, etc.).
- División entera y resto.
- Potencias (operador `**` y función `pow()`).
- Operaciones combinadas (jerarquía de operaciones).
- Operaciones con signos (manejo de números negativos).
- Conversión de tipos (usando `int()` para entrada de usuario).
- Funciones útiles para enteros (`abs()`, `divmod()`, `max()`, `min()`).
- Uso de enteros en estructuras repetitivas (`range()`).
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE OPERACIONES TIPO CON NÚMEROS ENTEROS.

6. LECCIÓN: ELEVACIÓN A POTENCIA Y SUS OPERACIONES INVERSAS

- Operador de potencia en Python (`**`).
- Uso de la función integrada `pow()` (incluye módulo opcional).
- Potencias negativas (cálculo del inverso multiplicativo).
- Potencias fraccionarias o raíces (usando exponentes fraccionarios como `(1/2)`).
- Uso de la librería `math` para raíces (`math.sqrt()`, `math.pow()`).
- Jerarquía de operaciones (la potencia tiene mayor prioridad).
- Raíces de números negativos (requiere la librería `cmath`).
- Ejemplo práctico: energía y potencia.
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE ELEVACIÓN A POTENCIA Y SUS OPERACIONES INVERSAS.

7. LECCIÓN: NÚMEROS PRIMOS, COMPUESTOS, MÚLTIPLOS Y DIVISORES

- Números primos y compuestos (definiciones).
- Verificar si un número es primo (comprobando divisibilidad hasta la raíz cuadrada).
- Listar los números primos hasta un límite.
- Múltiplos (si A % B == 0).
- Divisores (hallándolos con bucles).
- Contar el número de divisores.
- Clasificación usando divisores (primo si tiene 2, compuesto si tiene más).
- Optimización (probar solo hasta la raíz cuadrada).
- Uso de listas por comprensión para generar divisores.
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE NÚMEROS PRIMOS Y COMPUESTOS, MÚLTIPLOS Y DIVISORES.

8. LECCIÓN: MÁXIMO COMÚN DIVISOR (MCD)

- Introducción (definición y usos).
- Uso de `math.gcd()`.
- Método de Euclides (algoritmo iterativo).
- MCD de más de dos números (uso iterativo de `math.gcd()`).
- MCD con función personalizada.
- Propiedad: MCD(a, b) * MCM(a, b) = a * b.
- MCD con recursión (implementación de Euclides).
- MCD y fracciones (simplificación).
- MCD de lista de números (`functools.reduce`).
- Ejemplos prácticos (números coprimos).
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE MÁXIMO COMÚN DIVISOR.

9. LECCIÓN: MÍNIMO COMÚN MÚLTIPLO (MCM)

- Introducción (definición y usos).
- Relación entre MCD y MCM (MCM = (a * b) // MCD).
- Cálculo del MCM de dos números (método usando bucle `while True`).
- MCM de más de dos números (propiedad asociativa).
- Función personalizada de MCM.
- MCM y fracciones (denominadores comunes).
- MCM usando `reduce` para listas.
- Ejemplos prácticos (sincronización de eventos).
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE MÍNIMO COMÚN MÚLTIPLO.

10. LECCIÓN: PRINCIPIOS FUNDAMENTALES DE LA DIVISIBILIDAD

- Concepto de divisibilidad (resto cero usando %).
- Divisores y múltiplos (cómo obtenerlos en Python).
- Reglas de divisibilidad (por 2, 3, 5, 10).
- Divisores comunes y múltiplos comunes (MCD y MCM).
- Números coprimos (MCD = 1).
- Comprobación de divisibilidad en listas.

- Función general de divisibilidad.
- Relación entre divisibilidad y primos/compuestos.
- Suma de divisores y propiedades (números perfectos).
- Aplicaciones prácticas.
- PROGRAMAS RESUELTOS DE PRINCIPIOS FUNDAMENTALES DE LA DIVISIBILIDAD.

11. LECCIÓN: NÚMEROS FRACCIONARIOS Y SUS PROPIEDADES

- Introducción (numerador/denominador).
- Crear fracciones con `fractions.Fraction` (para precisión exacta).
- Operaciones básicas: Suma, Resta, Multiplicación, División.
- Simplificación automática.
- Conversión entre fracciones y `float`.
- Propiedades generales (conmutativa, asociativa, distributiva, identidad).
- Operaciones con listas de fracciones.
- Comparación de fracciones.
- Funciones útiles (`f.numerator`, `f.denominator`, `f.limit_denominator()`).
- Fracciones y decimales (uso mixto).
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE NÚMEROS FRACCIONARIOS.

12. LECCIÓN: REDUCCIÓN Y SIMPLIFICACIÓN DE QUEBRADOS

- Introducción (definición y forma más simple).
- Crear quebrados con `fractions.Fraction` (simplificación automática).
- Simplificación manual usando MCD (`math.gcd()`).
- Suma, Resta, Multiplicación y División de quebrados y su simplificación.
- Conversión entre quebrado y `float`.
- Crear funciones de simplificación.
- Reducción de lista de quebrados.
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE REDUCCIÓN Y SIMPLIFICACIÓN DE QUEBRADOS.

13. LECCIÓN: OPERACIONES CON NÚMEROS FRACCIONARIOS

- Crear fracciones.
- Suma de fracciones (cálculo exacto).
- Resta, Multiplicación y División de fracciones.
- Conversión entre fracciones y decimales.
- Simplificación automática.
- Propiedades importantes (Conmutativa, Asociativa, Distributiva, Identidad).
- Operaciones con listas de fracciones (`sum()`, bucles).
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE OPERACIONES CON FRACCIONES.

14. LECCIÓN: PROBLEMAS TIPO SOBRE QUEBRADOS COMUNES

- Introducción (usos: reparto, suma de partes, comparación).

- Crear quebrados comunes con `fractions.Fraction`.
- Suma, Resta, Multiplicación y División.
- Conversión entre quebrados y decimales.
- Problemas comunes: reparto de objetos, suma de partes, comparación.
- Funciones para quebrados.
- Resolver lista de quebrados.
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE PROBLEMAS TIPO SOBRE QUEBRADOS COMUNES.

15. LECCIÓN: FRACCIONES CONTINUAS

- Introducción (representación de números irracionales).
- Crear fracción continua de un número racional.
- Convertir fracción continua a número (reconstrucción).
- Conversión de número decimal a fracción continua.
- Propiedades importantes.
- Suma, Resta, Multiplicación y División usando el valor de la fracción exacta.
- Simplificación automática.
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE FRACCIONES CONTINUAS.

16. LECCIÓN: FRACCIONES DECIMALES

- Introducción (denominador potencia de 10).
- Crear fracciones decimales con `Fraction(decimal)`.
- Conversión de decimal a fracción.
- Operaciones con fracciones decimales (exactitud).
- Convertir fracción a decimal (`float()`).
- Simplificación automática.
- Funciones para fracciones decimales.
- Uso de listas de fracciones decimales.
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE FRACCIONES DECIMALES.

17. LECCIÓN: CONVERSIÓN DE FRACCIONES

- Introducción (Decimal ↔ Fracción, Impropia ↔ Mixta).
- Convertir decimal a fracción.
- Convertir fracción a decimal.
- Convertir fracción impropia a número mixto.
- Convertir número mixto a fracción impropia.
- Suma y resta con conversión automática.
- Multiplicación y división con fracciones.
- Uso de funciones para conversión.
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE CONVERSIÓN DE FRACCIONES.

18. LECCIÓN: SISTEMA MÉTRICO DECIMAL

- Introducción (sistema basado en múltiplos de 10).

- Concepto de unidades métricas (Longitud, Masa, Capacidad).
- Conversión entre unidades (multiplicar o dividir por potencias de 10).
- Funciones para conversiones.
- Conversiones con diccionarios.
- Operaciones con unidades métricas (convertir a unidad común).
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE SISTEMA MÉTRICO DECIMAL.

19. LECCIÓN: DENSIDAD

- Introducción (Densidad = Masa / Volumen).
- Calcular Densidad.
- Calcular Masa (Densidad * Volumen).
- Calcular Volumen (Masa / Densidad).
- Funciones para densidad.
- Conversión de unidades (g a kg, cm^3 a m^3).
- Operaciones con densidad (promedio).
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE DENSIDAD.

20. LECCIÓN: OTROS SISTEMAS DE MEDICIÓN

- Introducción (Sistema Inglés/Imperial, Temperatura, Tiempo, Área).
- Conversiones de longitud (Imperial ↔ Métrico).
- Conversiones de masa (Imperial ↔ Métrico).
- Conversiones de volumen.
- Conversiones de temperatura (Celsius ↔ Fahrenheit, Celsius ↔ Kelvin).
- Conversiones de área.
- Funciones para conversiones automáticas.
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE OTROS SISTEMAS DE MEDICIÓN.

21. LECCIÓN: ÁREAS DE FIGURAS PLANAS Y VOLÚMENES DE CUERPOS GEOMÉTRICOS

- Cálculo de Áreas de figuras planas (Cuadrado, Rectángulo, Triángulo, Círculo, Trapecio).
- Cálculo de Volúmenes de cuerpos geométricos (Cubo, Prisma rectangular, Cilindro, Esfera, Cono).
- Uso de `math.pi`.
- Funciones para áreas y volúmenes.
- 20 PROGRAMAS NUMERADOS Y RESUELTOS: ÁREAS Y VOLÚMENES.

22. LECCIÓN: TEOREMA DE PITÁGORAS

- Introducción (Fórmula general: $c^2 = a^2 + b^2$).
- Cálculo de la hipotenusa (`math.sqrt(a**2 + b**2)`).
- Cálculo de un cateto.
- Comprobación del teorema (verificar si es triángulo rectángulo).
- Uso de funciones para generalizar el cálculo.
- Aplicación geométrica (cálculo de distancias euclidianas).

- Redondeo de resultados.
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE TEOREMA DE PITÁGORAS.

23. LECCIÓN: LONGITUD Y TIEMPO

- Conceptos de Longitud (Métrico e Imperial y sus relaciones).
- Conversión de Longitud (ejemplos Métrico e Imperial).
- Funciones para Longitud.
- Conceptos de Tiempo (segundos, minutos, horas y sus relaciones).
- Conversiones de Tiempo.
- Funciones para Tiempo.
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE LONGITUD Y TIEMPO.

24. LECCIÓN: RAZONES Y PROPORCIONES

- Concepto de Razón (comparación por división).
- Proporciones (igualdad de dos razones, resolución por producto cruzado o regla de tres).
- Funciones para razones y proporciones.
- Uso de decimales y fracciones (`fractions.Fraction`).
- Aplicaciones prácticas (ajustar recetas, escalas).
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE RAZONES Y PROPORCIONES.

25. LECCIÓN: INTERÉS SIMPLE

- Fórmula del Interés Simple (I = C × r × t).
- Calcular Capital a partir del interés (C = I / (r × t)).
- Calcular Tiempo a partir del interés (t = I / (C × r)).
- Funciones para interés simple.
- Calcular Monto Total (M = C + I).
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE INTERÉS SIMPLE.

26. LECCIÓN: INTERÉS COMPUESTO

- Fórmula principal del Monto Compuesto (M = C × (1 + r)^t).
- Cálculo del Interés Compuesto (I = M - C).
- Calcular Capital Inicial a partir del monto.
- Calcular Tiempo a partir del monto (usando `math.log`).
- Funciones para interés compuesto.
- Monto total con interés compuesto mensual (si los períodos son subanuales).
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE INTERÉS COMPUESTO.

27. LECCIÓN: AMORTIZACIÓN

- Introducción (distribución de pagos de una deuda).
- Amortización Sistema Francés (Cuota Fija).
    - Fórmula de la cuota fija.
    - Cálculo de amortización mensual con ciclo (`for` loop).
    - Funciones para cuota fija y tabla de amortización.
- Amortización Sistema Decreciente (Capital amortizado constante).

- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE AMORTIZACIÓN.

28. LECCIÓN: DESCUENTO

- Fórmula del Descuento (D = P × d, PF = P - D).
- Calcular descuento en porcentaje.
- Funciones para descuento (monto, precio final, tasa).
- Descuento múltiple (descuentos sucesivos).
- Descuento en lista de productos (uso de listas por comprensión).
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE DESCUENTO.

29. LECCIÓN: REPARTO PROPORCIONAL

- Fórmula del Reparto Proporcional (Pi = M × (vi / S)).
- Implementación con listas.
- Funciones para reparto proporcional.
- Reparto con nombres (usando `zip`).
- Reparto redondeado.
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE REPARTO PROPORCIONAL.

30. LECCIÓN: REGLA DE COMPAÑÍA

- Almacenamiento de datos de empleados (diccionarios y listas).
- Calcular bonificación por ventas.
- Funciones para gestión de empleados.
- Suma de salarios.
- Reparto de utilidades (proporcional a ventas).
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE COMPAÑÍA.

31. LECCIÓN: ALIGACIÓN O MEZCLA

- Fórmula general de Aligación (C1 × V1 + C2 × V2 = Cf × Vf).
- Cálculo de volumen de cada componente (V1, V2).
- Ejemplo básico.
- Funciones para aligación.
- Regla de la aligación (reparto proporcional por diferencias).
- Aligación con más de dos componentes.
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE ALIGACIÓN.

32. LECCIÓN: ALEACIONES

- Fórmula general de Aleación (Masa del metal = M × (C / 100)).
- Fórmula para mezcla de dos aleaciones (M1C1 + M2C2 = MfCf).
- Cálculo de masa de cada componente (M1, M2).
- Funciones para aleaciones.
- Regla de mezcla proporcional.
- Aleación con varios componentes.
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE ALEACIONES.

33. LECCIÓN: PROMEDIOS

- Promedio Aritmético (uso de `sum()` y `len()`).
- Funciones para promedio.
- Promedio Ponderado.
- Promedio con datos ingresados por usuario.
- Promedio de diccionarios.
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE PROMEDIOS.

34. LECCIÓN: MONEDA

- Representar moneda en Python (uso de `decimal.Decimal` para precisión).
- Suma y Resta de moneda.
- Multiplicación y División.
- Redondeo de moneda.
- Conversión de moneda.
- Listas y diccionarios de moneda.
- Funciones para operaciones de moneda.
- Tipos de cambio de monedas importantes.
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE MONEDA.

35. LECCIÓN: REGLA CONJUNTA

- Introducción (proporciones múltiples o combinadas).
- Problemas básicos de regla conjunta (ejemplo de producción de máquinas).
- Función general para regla conjunta.
- Problemas con mezclas de cantidades.
- Problemas con tiempo y trabajo.
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE REGLA CONJUNTA.

36. LECCIÓN: SEGUROS

- Cálculo de Prima Básica (Prima = valor asegurado * tasa).
- Prima Anualizada.
- Indemnización por siniestro (cálculo de porcentaje).
- Seguro con Franquicia.
- Distribución de primas en grupo.
- Función para calcular prima.
- Seguros con años de vigencia.
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE SEGUROS.

37. LECCIÓN: ESTADÍSTICA

- Introducción (recolección, análisis, interpretación de datos).
- Listas de datos.
- Media (Promedio).
- Mediana (conteo par e impar).
- Moda (`collections.Counter`).
- Rango.

- Varianza y Desviación Estándar (`statistics` module).
- Frecuencia Relativa.
- Representación con listas y diccionarios.
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE ESTADÍSTICA.

38. LECCIÓN: OPERACIONES BINARIAS

- Representación binaria en Python (prefijo `0b`).
- Conversión entre decimal y binario (`bin()`, `int()`).
- Suma y Resta binaria.
- Multiplicación y División binaria.
- Operaciones Bit a Bit (`&` AND, `|` OR, `^` XOR, `~` NOT).
- Desplazamiento de Bits (`<<` izquierda, `>>` derecha).
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE OPERACIONES BINARIAS.

39. LECCIÓN: COORDENADAS

- Representación de coordenadas (2D y 3D).
- Acceso a los valores de las coordenadas.
- Distancia entre dos puntos 2D (fórmula de Pitágoras/Euclidiana).
- Distancia entre dos puntos 3D.
- Traslación de un punto.
- Escalamiento de un punto.
- Rotación de un punto en 2D (uso de trigonometría y `math.radians`).
- Listas de puntos y operaciones.
- 20 PROGRAMAS NUMERADOS Y RESUELTOS DE COORDENADAS.

1. LECCIÓN: SISTEMAS DE NUMERACIÓN

Introducción a los sistemas numéricos

En computación, los datos numéricos se representan utilizando diferentes **sistemas de numeración**. Los más comunes son:

- **Sistema decimal (base 10)**: utiliza los dígitos del 0 al 9.
- **Sistema binario (base 2)**: utiliza los dígitos 0 y 1.
- **Sistema octal (base 8)**: utiliza los dígitos del 0 al 7.
- **Sistema hexadecimal (base 16)**: utiliza los dígitos del 0 al 9 y las letras A, B, C, D, E, F.

Conversión entre sistemas numéricos

Python permite trabajar fácilmente con diferentes bases usando funciones integradas:

- **bin()** convierte un número decimal a binario.
- **oct()** convierte un número decimal a octal.
- **hex()** convierte un número decimal a hexadecimal.
- **int(x, base)** convierte un número de otra base a decimal.

Ejemplo:

```
n = 25
print(bin(n))   # binario
print(oct(n))   # octal
print(hex(n))   # hexadecimal
```

Resultado:

```
0b11001
0o31
0x19
```

El prefijo **0b** indica binario, **0o** indica octal y **0x** indica hexadecimal.

Conversión inversa (de otros sistemas a decimal)

Podemos convertir cualquier número a decimal usando la función **int()** y especificando su base.

Ejemplo:

```
binario = '11001'
decimal = int(binario, 2)
print(decimal)
```

Resultado:

```
25
```

También funciona con octal y hexadecimal:

```
octal = '31'
hexa = '19'
print(int(octal, 8))
print(int(hexa, 16))
```

Resultado:

```
25
25
```

Formato de salida personalizada

Podemos mostrar valores en diferentes bases sin los prefijos usando **format()** o f-strings.

Ejemplo:

```
n = 255
print(format(n, 'b'))  # binario
print(format(n, 'o'))  # octal
print(format(n, 'x'))  # hexadecimal
```

Resultado:

```
11111111
377
ff
```

También se puede usar **f"{n:b}"**, **f"{n:o}"**, **f"{n:x}"** para imprimir directamente.

Conversión de decimal a otras bases manualmente

Aunque Python lo hace automáticamente, entender la conversión manual es importante.

Ejemplo: convertir 13 a binario

1. 13 ÷ 2 = 6 resto 1
2. 6 ÷ 2 = 3 resto 0
3. 3 ÷ 2 = 1 resto 1
4. 1 ÷ 2 = 0 resto 1
   Lectura inversa → 1101

Operaciones con números binarios

Python permite realizar operaciones aritméticas entre valores binarios fácilmente convirtiéndolos a decimal o usando directamente enteros con prefijos 0b.

Ejemplo:

```
a = 0b1010
b = 0b1100
suma = a + b
print(bin(suma))
```

Resultado:

```
0b10110
```

Funciones útiles para bases numéricas

- **int()**: convierte cadenas a números.
- **bin(), oct(), hex()**: muestran valores en otras bases.
- **format()**: muestra valores con formato personalizado.
- **f-string**: también puede mostrar bases específicas.

Uso en la programación

Los sistemas de numeración son esenciales en la **representación de datos binarios**, **direcciones de memoria**, **valores de color en gráficos (hexadecimales)** y **códigos de caracteres**.

Por ejemplo, los colores HTML se representan en **hexadecimal**:

```
color_rojo = "#FF0000"
```

Y los valores ASCII se pueden convertir con **ord()** y **chr()**:

```
print(ord('A'))  # Decimal 65
print(hex(ord('A')))  # 0x41
```

Ejemplo completo de conversión múltiple

```
numero = 100
print("Decimal:", numero)
print("Binario:", bin(numero))
print("Octal:", oct(numero))
print("Hexadecimal:", hex(numero))
```

Resultado:

```
Decimal: 100
Binario: 0b1100100
Octal: 0o144
Hexadecimal: 0x64
```

Conversión inversa con input del usuario

```
valor = input("Introduce un número binario: ")
decimal = int(valor, 2)
print("Decimal equivalente:", decimal)
```

Comparación entre sistemas

Los valores se pueden comparar directamente si se convierten a decimal:

```
b1 = int('101', 2)
b2 = int('111', 2)
if b1 < b2:
    print("b1 es menor")
```

Resumen

- **binario (base 2)**: 0 y 1
- **octal (base 8)**: 0 a 7
- **decimal (base 10)**: 0 a 9
- **hexadecimal (base 16)**: 0–9 y A–F
- Funciones clave: **bin()**, **oct()**, **hex()**, **int()**, **format()**
- Conversión: **int(x, base)** para pasar de cualquier base a decimal
- Salida: **format(n, 'b'/'o'/'x')** para mostrar en otras bases

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE OTROS SISTEMAS DE NUMERACIÓN

1. Convertir número decimal a binario

```
n = int(input("Decimal: "))
print("Binario:", bin(n)[2:])
```

2. Convertir número decimal a octal

```
n = int(input("Decimal: "))
print("Octal:", oct(n)[2:])
```

3. Convertir número decimal a hexadecimal

```
n = int(input("Decimal: "))
print("Hexadecimal:", hex(n)[2:])
```

4. Convertir binario a decimal

```
b = input("Binario: ")
print("Decimal:", int(b, 2))
```

5. Convertir octal a decimal

```
o = input("Octal: ")
print("Decimal:", int(o, 8))
```

6. Convertir hexadecimal a decimal

```
h = input("Hexadecimal: ")
print("Decimal:", int(h, 16))
```

7. Mostrar el número en las cuatro bases

```
n = int(input("Número decimal: "))
print("Binario:", bin(n))
print("Octal:", oct(n))
print("Hexadecimal:", hex(n))
```

8. Convertir binario a hexadecimal

```
b = input("Binario: ")
decimal = int(b, 2)
print("Hexadecimal:", hex(decimal))
```

9. Convertir hexadecimal a binario

```
h = input("Hexadecimal: ")
decimal = int(h, 16)
print("Binario:", bin(decimal))
```

10. Sumar dos números binarios

```
a = input("Binario 1: ")
b = input("Binario 2: ")
resultado = int(a, 2) + int(b, 2)
print("Suma:", bin(resultado)[2:])
```

11. Restar dos números binarios

```
a = input("Binario 1: ")
b = input("Binario 2: ")
resultado = int(a, 2) - int(b, 2)
print("Resta:", bin(resultado)[2:])
```

12. Multiplicar dos números binarios

```
a = input("Binario 1: ")
b = input("Binario 2: ")
resultado = int(a, 2) * int(b, 2)
print("Multiplicación:", bin(resultado)[2:])
```

13. Convertir un número a todas las bases con format()

```
n = int(input("Número decimal: "))
print(format(n, 'b'), format(n, 'o'), format(n, 'x'))
```

14. Convertir una lista de números decimales a binario

```
nums = [1, 2, 3, 4, 5]
for n in nums:
    print(bin(n))
```

15. Ver tabla de equivalencias de 1 a 15

```
for i in range(1, 16):
    print(i, bin(i), oct(i), hex(i))
```

16. Convertir binario a octal

```
b = input("Binario: ")
decimal = int(b, 2)
print("Octal:", oct(decimal))
```

17. Convertir octal a binario

```
o = input("Octal: ")
decimal = int(o, 8)
print("Binario:", bin(decimal))
```

18.Mostrar representación hexadecimal de caracteres ASCII

```
for c in 'Python':
    print(c, hex(ord(c)))
```

19.Comparar dos números en diferentes bases

```
a = int(input("Binario: "), 2)
b = int(input("Hexadecimal: "), 16)
if a > b:
    print("El binario es mayor")
else:
    print("El hexadecimal es mayor o igual")
```

20.Convertir número decimal a bases personalizadas

```
def convertir(n, base):
    simbolos = "0123456789ABCDEF"
    resultado = ""
    while n > 0:
        resultado = simbolos[n % base] + resultado
        n //= base
    return resultado

n = int(input("Número decimal: "))
base = int(input("Base destino (2-16): "))
print("Resultado:", convertir(n, base))
```

---

2. LECCIÓN: NÚMEROS ROMANOS

Introducción a los números romanos

Los **números romanos** fueron el sistema de numeración usado en la antigua Roma y se basan en **letras mayúsculas** que representan valores fijos. A diferencia del sistema decimal, no utilizan posición ni ceros.

Símbolos principales:

I = 1
V = 5
X = 10
L = 50
C = 100
D = 500
M = 1000

Los números romanos se forman combinando estos símbolos según reglas específicas.

Reglas principales

1. Los números se suman cuando un símbolo menor está a la derecha de uno mayor (ejemplo: VI = 6).
2. Los números se restan cuando un símbolo menor está a la izquierda de uno mayor (ejemplo: IV = 4).
3. No se repite un símbolo más de tres veces seguidas (por ejemplo, III = 3, pero no IIII).
4. Los símbolos V, L y D **no se repiten**.
5. Los valores se construyen de mayor a menor de izquierda a derecha.

Ejemplos de números romanos comunes

1 = I
4 = IV
9 = IX
14 = XIV
40 = XL
90 = XC
400 = CD
900 = CM
2025 = MMXXV

Conversión de decimal a romano

Para convertir un número decimal a romano, se usa una **lista de valores decrecientes** y sus símbolos correspondientes.

El método consiste en restar el valor más alto posible tantas veces como quepa en el número, agregando su símbolo romano.

Ejemplo:

```
valores = [1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1]

simbolos = ["M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I"]


def decimal_a_romano(n):
    resultado = ""
    for i in range(len(valores)):
        while n >= valores[i]:
            resultado += simbolos[i]
            n -= valores[i]
    return resultado


print(decimal_a_romano(1987))
```

Resultado:
MCMLXXXVII

Conversión de romano a decimal

Para convertir de romano a decimal, se recorre la cadena de izquierda a derecha.
Si un símbolo tiene un valor menor que el siguiente, se resta; de lo contrario, se suma.

Ejemplo:

```
def romano_a_decimal(romano):
    valores = {'I':1,'V':5,'X':10,'L':50,'C':100,'D':500,'M':1000}
    total = 0
    for i in range(len(romano)):
        if i+1 < len(romano) and valores[romano[i]] < valores[romano[i+1]]:
            total -= valores[romano[i]]
        else:
            total += valores[romano[i]]
    return total


print(romano_a_decimal("MCMLXXXVII"))
```

Resultado:
1987

Combinando ambas conversiones

Podemos crear una función doble que reconozca si el valor es un número o un romano, y lo convierta automáticamente.

Ejemplo:

```
def convertir(valor):
    if valor.isdigit():
        return decimal_a_romano(int(valor))
    else:
        return str(romano_a_decimal(valor))


print(convertir("2025"))
print(convertir("MMXXV"))
```

Resultado:
MMXXV
2025

Errores comunes

- Usar más de tres símbolos iguales seguidos (por ejemplo, IIII)
- Colocar símbolos incorrectos en orden (por ejemplo, IC para 99 en vez de XCIX)
- No respetar las reglas de resta y suma.

Python puede ayudar a **validar** si una cadena romana es correcta.

Validación de número romano

Podemos usar expresiones regulares para verificar si un número romano es válido:

```
import re
patron = "^M{0,4}(CM|CD|D?C{0,3})(XC|XL|L?X{0,3})(IX|IV|V?I{0,3})$"
romano = input("Introduce número romano: ")
if re.match(patron, romano):
    print("Número romano válido")
else:
    print("Número romano inválido")
```

Aplicaciones prácticas

Los números romanos se usan en:

- Relojes antiguos (por ejemplo, XII para 12)

- Nombres de reyes (por ejemplo, Felipe VI)
- Enumeraciones en libros o películas (Capítulo IV, Episodio V)
- Fechas en monumentos (por ejemplo, MDCCCLXXXV = 1885)

Resumen de la lección

- Los **números romanos** usan letras en lugar de cifras.
- Las letras tienen valores fijos y se combinan por reglas de suma o resta.
- En Python, se puede **convertir fácilmente entre decimal y romano** usando bucles o diccionarios.
- Se puede **validar** si una cadena cumple las reglas del sistema romano.
- Comprender los números romanos ayuda a entender cómo representar valores fuera del sistema posicional.

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE NÚMEROS ROMANOS

1. Convertir número decimal a romano

```
def decimal_a_romano(n):
    valores = [1000,900,500,400,100,90,50,40,10,9,5,4,1]
    simbolos = ["M","CM","D","CD","C","XC","L","XL","X","IX","V","IV","I"]
    resultado = ""
    for i in range(len(valores)):
        while n >= valores[i]:
            resultado += simbolos[i]
            n -= valores[i]
    return resultado
print(decimal_a_romano(2025))
```

2. Convertir número romano a decimal

```
def romano_a_decimal(romano):
    valores = {'I':1,'V':5,'X':10,'L':50,'C':100,'D':500,'M':1000}
    total = 0
    for i in range(len(romano)):
        if i+1 < len(romano) and valores[romano[i]] < valores[romano[i+1]]:
            total -= valores[romano[i]]
        else:
            total += valores[romano[i]]
    return total
```

```
print(romano_a_decimal("MMXXV"))
```

3. Mostrar tabla de equivalencias del 1 al 20

```
for i in range(1,21):
    print(i, decimal_a_romano(i))
```

4. Validar si un número romano es correcto

```
import re
r = input("Número romano: ")
if re.match("^M{0,4}(CM|CD|D?C{0,3})(XC|XL|L?X{0,3})(IX|IV|V?I{0,3})$", r):
    print("Válido")
else:
    print("Inválido")
```

5. Convertir lista de números decimales a romanos

```
lista = [4,9,14,29,44]
print([decimal_a_romano(x) for x in lista])
```

6. Convertir lista de números romanos a decimales

```
romanos = ["X", "XL", "XC", "C"]
print([romano_a_decimal(r) for r in romanos])
```

7. Pedir un número y mostrar sus representaciones romana y decimal

```
valor = input("Introduce número o romano: ")
if valor.isdigit():
    print(decimal_a_romano(int(valor)))
else:
    print(romano_a_decimal(valor))
```

8. Mostrar equivalencias de siglos del I al XXI

```
for i in range(1,22):
    print("Siglo", decimal_a_romano(i))
```

9. Comparar dos números romanos

```
a = "XIV"
b = "XVI"
if romano_a_decimal(a) > romano_a_decimal(b):
    print(a, "es mayor")
else:
    print(b, "es mayor")
```

10. Sumar dos números romanos

```
a = "XII"
```

```
b = "VIII"
suma = romano_a_decimal(a) + romano_a_decimal(b)
print(decimal_a_romano(suma))
```

11.Restar dos números romanos

```
a = "XX"
b = "V"
resta = romano_a_decimal(a) - romano_a_decimal(b)
print(decimal_a_romano(resta))
```

12.Multiplicar dos números romanos

```
a = "X"
b = "VI"
mul = romano_a_decimal(a) * romano_a_decimal(b)
print(decimal_a_romano(mul))
```

13.Convertir fecha en romano

```
dia = 13
mes = 11
anio = 2025
print(decimal_a_romano(dia), decimal_a_romano(mes), decimal_a_romano(anio))
```

14.Mostrar todos los números romanos del 50 al 60

```
for i in range(50,61):
    print(i, decimal_a_romano(i))
```

15.Mostrar año actual en romano

```
import datetime
a = datetime.datetime.now().year
print(decimal_a_romano(a))
```

16.Verificar si una cadena contiene solo símbolos válidos

```
r = input("Cadena: ")
if all(c in "IVXLCDM" for c in r):
    print("Contiene solo símbolos válidos")
else:
    print("Contiene caracteres no válidos")
```

17.Convertir número romano en letras a mayúsculas antes de procesar

```
r = input("Número romano: ").upper()
print(romano_a_decimal(r))
```

18.Calcular el doble de un número romano

```
r = "XIV"
doble = romano_a_decimal(r)*2
print(decimal_a_romano(doble))
```

19. Convertir secuencia mixta de números

```
datos = ["IV", "9", "XII", "50"]
for d in datos:
    if d.isdigit():
        print(decimal_a_romano(int(d)))
    else:
        print(romano_a_decimal(d))
```

20. Crear función general de conversión automática

```
def convertir(valor):
    if valor.isdigit():
        return decimal_a_romano(int(valor))
    else:
        return str(romano_a_decimal(valor))
print(convertir("39"))
print(convertir("XLII"))
```

---

3. LECCIÓN: RELACIONES DE IGUALDAD Y DESIGUALDAD

Introducción

En programación, las **relaciones de igualdad y desigualdad** permiten comparar valores entre sí. Estas comparaciones devuelven un valor **booleano** (True o False), y son fundamentales en **estructuras condicionales** y **bucles**.

En Python, los operadores relacionales se usan para comparar **números, cadenas, caracteres o expresiones**.

Operadores relacionales principales

Los operadores más utilizados son:

== igual que
!= distinto de
> mayor que
< menor que
>= mayor o igual que
<= menor o igual que

Cada uno devuelve un **resultado booleano**, es decir, **True o False**.

Ejemplo básico

```
a = 10
b = 5
print(a == b)  # False
print(a != b)  # True
print(a > b)   # True
print(a < b)   # False
```

En este ejemplo, Python evalúa las relaciones y devuelve valores lógicos según la comparación.

Importancia de los valores booleanos

Los valores **True** y **False** son esenciales en **estructuras condicionales** como if, elif y else.

Ejemplo:

```
x = 8
if x == 8:
    print("El valor es igual a 8")
else:
    print("El valor no es igual a 8")
```

Salida:
El valor es igual a 8

Comparación entre distintos tipos de datos

Python permite comparar diferentes tipos compatibles, como números o cadenas.

1. Comparando números:

```
a = 7
b = 3
print(a > b)  # True
```

2. Comparando cadenas:

```
palabra1 = "casa"
palabra2 = "perro"
print(palabra1 == palabra2)  # False
```

Las cadenas se comparan **alfabéticamente según el código ASCII**, por eso "casa" < "perro".

Igualdad con variables booleanas

```
estado = True
if estado == True:
    print("Está activo")
else:
    print("Está inactivo")
```

También se puede escribir más simple:

```
if estado:
    print("Activo")
```

Combinación de relaciones

Se pueden combinar comparaciones usando **and**, **or** y **not**.

Ejemplo:

```
edad = 25
print(edad > 18 and edad < 30)  # True
print(edad < 18 or edad > 60)   # False
print(not(edad == 25))          # False
```

Operaciones encadenadas

Python permite encadenar comparaciones múltiples en una sola expresión.

Ejemplo:

```
x = 10
print(5 < x < 15)  # True
print(5 < x > 15)  # False
```

Esto se evalúa como (5 < x) and (x < 15).

Comparaciones entre expresiones

Las comparaciones también pueden incluir **operaciones aritméticas**:

```
a = 5
b = 10
print(a * 2 == b)  # True
print(a + b != 20) # True
```

Comparaciones de flotantes

Los números decimales pueden dar resultados inesperados por la **precisión binaria**.

```
a = 0.1 + 0.2
print(a == 0.3)  # False por imprecisión de punto flotante
```

Para solucionar esto, se recomienda usar la función **round()** o comparar con tolerancia:

```
print(round(a,1) == 0.3)  # True
```

Comparaciones de texto con mayúsculas y minúsculas

Python distingue entre mayúsculas y minúsculas:

```
print("Hola" == "hola")  # False
print("Hola".lower() == "hola")  # True
```

Uso en decisiones lógicas

Las comparaciones permiten tomar decisiones:

```
nota = 7
if nota >= 5:
    print("Aprobado")
else:
    print("Reprobado")
```

Comparaciones de objetos

Cuando se comparan listas, tuplas o cadenas, Python las evalúa **elemento por elemento**:

```
print([1,2,3] == [1,2,3])  # True
print((1,2) < (2,1))       # True porque 1 < 2
```

Operadores “is” y “is not”

Aunque no son operadores de igualdad de valor, comparan **identidad de objeto**:

```
a = [1,2]
b = a
c = [1,2]
print(a == b)      # True
print(a is b)      # True (misma referencia)
print(a == c)      # True
print(a is c)      # False (diferentes objetos)
```

Conclusión

Las **relaciones de igualdad y desigualdad** son la base de la toma de decisiones en Python.
Permiten comparar datos numéricos, de texto, booleanos u objetos, y son esenciales en condiciones, bucles y validaciones.
Comprender cómo funcionan ayuda a construir programas **lógicos, precisos y confiables**.

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE RELACIONES DE IGUALDAD Y DESIGUALDAD

1. Verificar si dos números son iguales

```
a = 10
b = 10
print(a == b)
```

2. Verificar si dos números son diferentes

```
a = 8
b = 5
print(a != b)
```

3. Comparar si un número es mayor que otro

```
a = 12
b = 9
print(a > b)
```

4. Comparar si un número es menor o igual que otro

```
x = 7
y = 7
```

```
print(x <= y)
```

5. Evaluar una edad para mayoría de edad

```
edad = 18
if edad >= 18:
    print("Mayor de edad")
```

6. Comprobar si una palabra es igual a otra

```
p1 = "Python"
p2 = "python"
print(p1 == p2)
```

7. Comparar cadenas ignorando mayúsculas

```
p1 = "Hola"
p2 = "hola"
print(p1.lower() == p2.lower())
```

8. Comprobar si dos listas son iguales

```
a = [1,2,3]
b = [1,2,3]
print(a == b)
```

9. Verificar desigualdad entre listas

```
a = [1,2]
b = [2,1]
print(a != b)
```

10. Comprobar rango con comparación encadenada

```
x = 10
print(5 < x < 15)
```

11. Determinar si una persona tiene edad entre 20 y 30

```
edad = 25
if 20 <= edad <= 30:
    print("Edad entre 20 y 30")
```

12. Verificar igualdad de flotantes con redondeo

```
a = 0.1 + 0.2
print(round(a,1) == 0.3)
```

13. Comparar resultados de operaciones

```
a = 5
b = 10
print(a * 2 == b)
```

14.Validar contraseña

```
clave = input("Introduce contraseña: ")
print(clave == "1234")
```

15.Comprobar si dos variables apuntan al mismo objeto

```
a = [1,2,3]
b = a
print(a is b)
```

16.Verificar si dos objetos son diferentes

```
a = [1,2]
b = [1,2]
print(a is not b)
```

17.Comparar dos caracteres

```
c1 = 'a'
c2 = 'b'
print(c1 < c2)
```

18.Usar comparación dentro de un if

```
x = 50
if x != 0:
    print("El número no es cero")
```

19.Verificar si una palabra tiene una longitud específica

```
palabra = "python"
if len(palabra) == 6:
    print("Longitud correcta")
```

20.Comparar tres números y mostrar el mayor

```
a, b, c = 5, 9, 7
if a >= b and a >= c:
    print(a, "es mayor")
elif b >= a and b >= c:
    print(b, "es mayor")
else:
    print(c, "es mayor")
```

---

4. LECCIÓN: OPERACIONES ARITMÉTICAS

Introducción

En Python, las **operaciones aritméticas** son la base de cualquier cálculo numérico. Se realizan con **números enteros o decimales**, y utilizan **símbolos especiales** llamados **operadores**.

Los operadores aritméticos más usados son:

`+` → suma
`-` → resta
*** → multiplicación
/ → división
// → división entera
% → resto o módulo
**** → potencia

---

1. La suma (+)

La **suma** permite añadir dos o más valores.
Ejemplo:

```
a = 10
b = 5
resultado = a + b
print("Suma:", resultado)
```

Salida:
Suma: 15

También se pueden sumar variables o expresiones:

```
x = 3 + 7 + 2
print(x)
```

Resultado: 12

---

2. La resta (-)

La **resta** se utiliza para calcular la diferencia entre dos valores.

```
a = 15
b = 8
print("Resta:", a - b)
```

Resultado: 7

La resta también puede dar números negativos:

```
print(5 - 9)
```

Resultado: -4

---

3. La multiplicación (*)

La **multiplicación** usa el símbolo * (asterisco).

```
a = 4
b = 6
print("Multiplicación:", a * b)
```

Resultado: 24

También se usa para repetir cadenas:

```
print("Hola" * 3)
```

Resultado: HolaHolaHola

---

4. La división (/)

La **división normal** devuelve un número decimal (flotante).

```
a = 10
b = 4
print("División:", a / b)
```

Resultado: 2.5

Siempre devuelve un **float**, incluso si es exacta.

---

5. La división entera (//)

La **división entera** descarta los decimales y devuelve solo la parte entera del cociente.

```
a = 10
b = 4
print("División entera:", a // b)
```

Resultado: 2

Sirve cuando solo interesa el número entero del resultado.

---

6. El módulo (%)

El operador **%** devuelve el **resto** de una división.

```
a = 10
b = 4
print("Resto:", a % b)
```

Resultado: 2

Ejemplo útil: saber si un número es par.

```
n = 8
if n % 2 == 0:
    print("Par")
else:
    print("Impar")
```

---

7. La potencia ()**

El operador de **potencia** calcula una base elevada a un exponente.

```
a = 3
b = 4
print("Potencia:", a ** b)
```

Resultado: 81

También se puede usar la función **pow()**:

```
print(pow(2, 5))
```

Resultado: 32

---

8. Raíces cuadradas y otras raíces

Las raíces se calculan elevando a una **potencia fraccionaria**.
Por ejemplo, la raíz cuadrada de x es **x elevado a (1/2)**.

```
n = 25
print("Raíz cuadrada:", n ** 0.5)
```

Resultado: 5.0

Raíz cúbica:

```
n = 27
print("Raíz cúbica:", n ** (1/3))
```

---

9. Prioridad de operaciones

Python sigue las **reglas matemáticas de precedencia**:

1. Paréntesis ( )
2. Potencias **
3. Multiplicación, división, división entera y módulo (*, /, //, %)
4. Suma y resta (+, -)

Ejemplo:

```
resultado = 5 + 2 * 3 ** 2
print(resultado)
```

Resultado: 23

---

10. Uso de paréntesis

Se pueden usar **paréntesis** para cambiar la prioridad:

```
resultado = (5 + 2) * 3 ** 2
print(resultado)
```

Resultado: 63

---

11. Operaciones combinadas

Python permite combinar varias operaciones en una sola expresión.

```
a = 10
b = 3
c = 2
resultado = (a + b) * c - (a // b)
print("Resultado:", resultado)
```

---

12. Funciones matemáticas adicionales

El módulo **math** incluye más operaciones:

```
import math
print(math.sqrt(49))     # raíz cuadrada
print(math.pow(2, 3))    # potencia
print(math.floor(7.8))   # redondear hacia abajo
print(math.ceil(7.2))    # redondear hacia arriba
```

---

13. Conversión de tipos numéricos

Se pueden convertir números flotantes a enteros y viceversa.

```
x = 7.9
print(int(x))
print(float(5))
```

---

14. Redondeo

La función **round()** redondea un número al entero más cercano.

```
print(round(7.6))
print(round(7.2))
```

---

15. Resumen general

- + suma
- - resta
- *** **multiplica
- / divide con decimales
- // divide enteramente
- **%** obtiene el resto
- **** calcula potencias
- **x ** (1/n)** calcula raíces

Python respeta la **jerarquía de operaciones**, y permite el uso de **math** para operaciones avanzadas.

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE OPERACIONES ARITMÉTICAS

1. Suma de dos números

```
a = int(input("A: "))
b = int(input("B: "))
print("Suma:", a + b)
```

2. Resta de dos números

```
a = int(input("A: "))
b = int(input("B: "))
print("Resta:", a - b)
```

3. Multiplicación de dos números

```
a = int(input("A: "))
b = int(input("B: "))
print("Producto:", a * b)
```

4. División de dos números

```
a = float(input("A: "))
b = float(input("B: "))
print("División:", a / b)
```

5. División entera de dos números

```
a = int(input("A: "))
b = int(input("B: "))
print("Cociente entero:", a // b)
```

6. Resto de una división

```
a = int(input("A: "))
b = int(input("B: "))
print("Resto:", a % b)
```

7. Potencia de un número

```
base = int(input("Base: "))
exp = int(input("Exponente: "))
print("Resultado:", base ** exp)
```

8. Raíz cuadrada de un número

```
n = float(input("Número: "))
print("Raíz cuadrada:", n ** 0.5)
```

9. Raíz cúbica de un número

```
n = float(input("Número: "))
print("Raíz cúbica:", n ** (1/3))
```

10. Suma de tres números

```
a = int(input("A: "))
b = int(input("B: "))
c = int(input("C: "))
print("Suma total:", a + b + c)
```

11. Calcular el promedio de tres números

```
a = int(input("A: "))
b = int(input("B: "))
c = int(input("C: "))
print("Promedio:", (a + b + c) / 3)
```

12. Calcular el cuadrado de un número

```
n = int(input("Número: "))
print("Cuadrado:", n ** 2)
```

13. Calcular el cubo de un número

```
n = int(input("Número: "))
print("Cubo:", n ** 3)
```

14. Calcular el área de un triángulo

```
base = float(input("Base: "))
altura = float(input("Altura: "))
print("Área:", (base * altura) / 2)
```

15. Calcular el perímetro de un cuadrado

```
lado = float(input("Lado: "))
```

```
print("Perímetro:", lado * 4)
```

16.Calcular el resto de dividir un número entre 7

```
n = int(input("Número: "))
print("Resto:", n % 7)
```

17.Calcular el doble y triple de un número

```
n = int(input("Número: "))
print("Doble:", n * 2)
print("Triple:", n * 3)
```

18.Calcular la potencia de 2 elevado a un número

```
n = int(input("Exponente: "))
print("2 elevado a", n, "es", 2 ** n)
```

19.Calcular la raíz cuarta de un número

```
n = float(input("Número: "))
print("Raíz cuarta:", n ** 0.25)
```

20.Calcular el resultado de una operación combinada

```
a = 5
b = 3
c = 2
resultado = (a + b) ** 2 / (c + 1)
print("Resultado:", resultado)
```

---

5. LECCIÓN: OPERACIONES CON NÚMEROS ENTEROS

Introducción

En Python, los **números enteros** pertenecen al tipo de dato **int**.
Representan cantidades **sin parte decimal**, y pueden ser **positivos, negativos o cero**.
Son fundamentales en todas las operaciones matemáticas y lógicas dentro de los programas.

Ejemplo:

```
x = 10
y = -3
z = 0
print(x, y, z)
```

Resultado:
10 -3 0

Características de los números enteros

1. Se escriben directamente, sin comillas.
2. Python soporta **enteros muy grandes**, sin límite fijo.
3. Se pueden usar en **operaciones aritméticas, condicionales y bucles**.
4. Se pueden combinar con variables o expresiones.

Ejemplo:

```
a = 12
b = 8
c = a * 2 - b
print(c)
```

Resultado: 16

Operadores aritméticos básicos

Python usa los mismos símbolos que las matemáticas:

`+` → suma
`-` → resta
*** → multiplicación
/ → división (resultado decimal)
// → división entera (descarta decimales)
% → módulo o resto de la división
**** → potencia o exponente

Ejemplo:

```
a = 15
```

```
b = 4
print(a + b)
print(a - b)
print(a * b)
print(a / b)
print(a // b)
print(a % b)
print(a ** b)
```

Resultado:
19
11
60
3.75
3
3
50625

División entera y resto

La **división entera** devuelve solo la parte entera del cociente.
El **módulo (%)** devuelve el resto o residuo de la división.

Ejemplo:

```
a = 17
b = 5
print("Cociente:", a // b)
print("Resto:", a % b)
```

Resultado:
Cociente: 3
Resto: 2

Potencias

El operador ****** eleva un número a otro.
Ejemplo:

```
base = 3
exponente = 4
print("Resultado:", base ** exponente)
```

Resultado:
81

También se puede usar la función **pow()**:

```
print(pow(2, 5))
```

Resultado:
32

Operaciones combinadas

Python sigue la **jerarquía de operaciones**:

1. Paréntesis ( )
2. Potencias
3. Multiplicaciones, divisiones, divisiones enteras y módulos
4. Sumas y restas

Ejemplo:

```
resultado = 5 + 2 * 3 ** 2
```

```
print(resultado)
```

Resultado: 23

Si cambiamos los paréntesis:

```
resultado = (5 + 2) * 3 ** 2
```

```
print(resultado)
```

Resultado: 63

Operaciones con signos

Los números negativos se escriben con **-** y respetan las reglas matemáticas.

(+)*(+) = +
(-)*(+) = -
(-)*(-) = +

Ejemplo:

```
a = -5
```

```
b = 3
```

```
print(a * b)
```

```
print(-a * -b)
```

Resultado:
-15
15

Conversión de tipos

Cuando se reciben datos por teclado con **input()**, se leen como cadenas.
Para convertirlos a enteros, se usa **int()**.

Ejemplo:

```
n = int(input("Introduce un número entero: "))
print("El doble es:", n * 2)
```

Funciones útiles para enteros

- **abs(x)** → valor absoluto
- **divmod(x, y)** → devuelve (cociente, resto)
- **max(x, y)** → mayor de dos números
- **min(x, y)** → menor de dos números

Ejemplo:

```
a = -9
b = 4
print(abs(a))
print(divmod(17, 5))
print(max(a, b))
print(min(a, b))
```

Resultado:
9
(3, 2)
4
-9

Uso de enteros en estructuras repetitivas

Los enteros son esenciales en los bucles **for** y **while**, ya que sirven para contar iteraciones.

Ejemplo:

```
for i in range(5):
    print(i)
```

Resultado:
0
1
2
3
4

Resumen general

- **int** representa números enteros.
- Se pueden sumar, restar, multiplicar, dividir, elevar, o calcular restos.
- Python usa **jerarquía de operaciones**.
- Se pueden combinar en expresiones complejas.
- Son fundamentales en **contadores, índices y estructuras de control**.

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE OPERACIONES TIPO CON NÚMEROS ENTEROS

1. Calcular el doble de un número entero

```
n = int(input("Número: "))
print("El doble es:", n * 2)
```

2. Calcular el triple de un número entero

```
n = int(input("Número: "))
print("El triple es:", n * 3)
```

3. Calcular la suma de dos enteros

```
a = int(input("A: "))
b = int(input("B: "))
print("Suma:", a + b)
```

4. Calcular la diferencia de dos números

```
a = int(input("A: "))
b = int(input("B: "))
print("Diferencia:", a - b)
```

5. Multiplicar dos enteros

```
a = int(input("A: "))
b = int(input("B: "))
print("Producto:", a * b)
```

6. Dividir dos números enteros (división normal)

```
a = int(input("A: "))
b = int(input("B: "))
print("División:", a / b)
```

7. Calcular la división entera y el resto

```
a = int(input("A: "))
b = int(input("B: "))
print("Cociente:", a // b)
print("Resto:", a % b)
```

8. Calcular el cuadrado de un número

```
n = int(input("Número: "))
print("Cuadrado:", n ** 2)
```

9. Calcular el cubo de un número

```
n = int(input("Número: "))
print("Cubo:", n ** 3)
```

10. Calcular la potencia de un número

```
base = int(input("Base: "))
exp = int(input("Exponente: "))
print("Resultado:", base ** exp)
```

11. Calcular la suma de los primeros n números enteros

```
n = int(input("n: "))
suma = n * (n + 1) // 2
print("Suma:", suma)
```

12. Calcular el promedio de tres números enteros

```
a = int(input("A: "))
b = int(input("B: "))
c = int(input("C: "))
print("Promedio:", (a + b + c) / 3)
```

13. Calcular el valor absoluto de un número

```
n = int(input("Número: "))
print("Valor absoluto:", abs(n))
```

14. Calcular el área de un rectángulo (base y altura enteras)

```
base = int(input("Base: "))
altura = int(input("Altura: "))
print("Área:", base * altura)
```

15. Calcular el perímetro de un cuadrado

```
lado = int(input("Lado: "))
print("Perímetro:", lado * 4)
```

16. Comprobar si un número entero es par o impar

```
n = int(input("Número: "))
if n % 2 == 0:
    print("Par")
else:
    print("Impar")
```

17.Calcular el cociente y resto de dividir 45 entre 6

```
a = 45
b = 6
print("Cociente:", a // b)
print("Resto:", a % b)
```

18.Calcular la diferencia absoluta entre dos números

```
a = int(input("A: "))
b = int(input("B: "))
print("Diferencia absoluta:", abs(a - b))
```

19.Elevar un número entero al cuadrado y restarle 10

```
n = int(input("Número: "))
print("Resultado:", n ** 2 - 10)
```

20.Calcular una expresión combinada con enteros

```
a = 8
b = 3
c = 2
resultado = (a + b) * c - (a // b) + (a % b)
print("Resultado:", resultado)
```

---

6. LECCIÓN: ELEVACIÓN A POTENCIA Y SUS OPERACIONES INVERSAS

Introducción

En Python, la **elevación a potencia** y las **operaciones inversas** como las **raíces** son operaciones matemáticas muy importantes que se utilizan en muchos campos: desde la física hasta la informática y la estadística.
La **potencia** es una forma abreviada de multiplicar un número por sí mismo varias veces.
Por ejemplo, **$3^2 = 3 \times 3 = 9$**, y **$5^3 = 5 \times 5 \times 5 = 125$**.

Operador de potencia en Python

En Python, la potencia se realiza con el **operador **** (doble asterisco).
La sintaxis es:

```
resultado = base ** exponente
```

Ejemplo:

```
a = 2
b = 5
print(a ** b)
```

Resultado:
32

En este ejemplo, **2 elevado a la quinta potencia es 32**.

Usando la función integrada pow()

Python también dispone de la **función pow()**, que puede utilizarse para calcular potencias.

```
resultado = pow(base, exponente)
```

Ejemplo:

```
print(pow(3, 4))
```

Resultado:
81

pow() también admite un **tercer argumento opcional** que calcula el resultado **módulo** de una operación.

```
print(pow(3, 4, 5))
```

Resultado:
1
(ya que $3^4 = 81$, y $81 \% 5 = 1$)

Potencias negativas

Cuando el exponente es negativo, Python calcula el **inverso multiplicativo** del número.
Ejemplo:

```
print(2 ** -3)
```

Resultado:
0.125
(ya que $2^{-3} = 1 / 2^3 = 1/8 = 0.125$)

Potencias fraccionarias (raíces)

Las **raíces** son las operaciones inversas de las potencias.
Por ejemplo, la **raíz cuadrada** de un número es el **número que elevado al cuadrado da el valor original**.

En Python se puede obtener una raíz utilizando un **exponente fraccionario**:

```
raiz_cuadrada = x ** (1/2)
```

```
raiz_cubica = x ** (1/3)
```

Ejemplo:

```
x = 16
```

```
print(x ** 0.5)
```

Resultado:
4.0

Otro ejemplo:

```
x = 27
```

```
print(x ** (1/3))
```

Resultado:
3.0

Usando la librería math para raíces

Python también incluye funciones para calcular raíces mediante la librería **math**.
Primero hay que importarla:

```
import math
```

Y luego se puede usar:

```
math.sqrt(x)   # raíz cuadrada
```

```
math.pow(x, y) # potencia
```

Ejemplo:

```
import math
```

```
print(math.sqrt(81))
```

Resultado:
9.0

Jerarquía de operaciones

En Python, las potencias tienen **mayor prioridad que la multiplicación, división, suma y resta**.
Ejemplo:

```
resultado = 2 + 3 ** 2 * 2

print(resultado)
```

Resultado:
20
Porque primero se calcula $3^2 = 9$, luego 9 * 2 = 18, y por último 2 + 18 = 20.

Combinación de potencias y raíces

Podemos mezclar potencias y raíces en una misma expresión.
Ejemplo:

```
x = 4

resultado = (x ** 3) ** 0.5

print(resultado)
```

Resultado:
8.0

(Se calcula $4^3 = 64$, y la raíz cuadrada de 64 es 8)

Raíces de números negativos

Python **no admite raíces pares de números negativos** con exponentes fraccionarios porque el resultado sería un número imaginario.
Ejemplo:

```
print((-8) ** (1/3))
```

Resultado:
nan (error o resultado no definido en algunos casos)

Para trabajar con números complejos se usa la librería **cmath**, que maneja raíces de negativos correctamente:

```
import cmath

print(cmath.sqrt(-9))
```

Resultado:
3j

Ejemplo práctico: energía y potencia

Supón que un motor tiene una potencia de 1000 W durante 3 horas. Si se eleva la potencia a la tercera parte del tiempo, podríamos modelar un comportamiento físico simplificado con potencias:

```
potencia = 1000
tiempo = 3
energia = potencia ** (1/tiempo)
print(energia)
```

Esto da una idea del concepto de **potencia fraccionaria** en modelos de crecimiento o desgaste.

Resumen general

- **El operador ** eleva una base a un exponente.**
- **pow()** es otra forma de calcular potencias y permite módulo.
- **Raíces** se obtienen usando exponentes fraccionarios o funciones de **math**.
- **Potencias negativas** producen valores inversos.
- **Raíces de negativos** requieren el módulo **cmath**.
- **La jerarquía de operaciones** prioriza potencias antes que multiplicaciones o sumas.

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE ELEVACIÓN A POTENCIA Y SUS OPERACIONES INVERSAS

1. Calcular el cuadrado de un número

```
n = int(input("Número: "))
print("Cuadrado:", n ** 2)
```

2. Calcular el cubo de un número

```
n = int(input("Número: "))
print("Cubo:", n ** 3)
```

3. Calcular una potencia con pow()

```
a = 4
b = 5
print(pow(a, b))
```

4. Calcular la raíz cuadrada usando **

```
x = int(input("Número: "))
print("Raíz cuadrada:", x ** 0.5)
```

5. Calcular la raíz cúbica de un número

```
x = int(input("Número: "))
print("Raíz cúbica:", x ** (1/3))
```

6. Calcular la cuarta potencia de un número

```
n = int(input("Número: "))
print("Cuarta potencia:", n ** 4)
```

7. Calcular la raíz cuarta

```
x = int(input("Número: "))
print("Raíz cuarta:", x ** (1/4))
```

8. Calcular potencia negativa

```
x = int(input("Número: "))
print("Inverso cúbico:", x ** -3)
```

9. Calcular una potencia y su raíz inversa

```
x = 9
p = x ** 2
r = p ** 0.5
print("Número:", x, " Potencia:", p, " Raíz inversa:", r)
```

10. Usar math.sqrt()

```
import math
x = 64
print(math.sqrt(x))
```

11. Elevar 2 a la 10 y mostrar el resultado

```
print(2 ** 10)
```

12. Calcular una potencia con exponente decimal

```
x = 9
y = 1.5
print(x ** y)
```

13. Calcular raíz de un número negativo con cmath

```
import cmath
print(cmath.sqrt(-25))
```

14. Calcular el área de un círculo usando potencia

```
import math
r = int(input("Radio: "))
area = math.pi * (r ** 2)
print("Área:", area)
```

15. Calcular una potencia y restarle 10

```
n = int(input("Número: "))
print("Resultado:", (n ** 3) - 10)
```

16. Calcular una expresión combinada

```
a = 2
b = 3
c = 4
resultado = (a ** b) + (b ** c)
print(resultado)
```

17.Comparar dos potencias

```
a = 2 ** 5
b = 3 ** 3
print(a > b)
```

18.Calcular raíz cúbica exacta

```
x = 125
print(x ** (1/3))
```

19.Calcular potencia modular

```
print(pow(7, 4, 5))
```

20.Calcular la raíz cuadrada de la suma de dos potencias

```
x = 3
y = 4
resultado = (x**2 + y**2) ** 0.5
print("Resultado:", resultado)
```

---

7. LECCIÓN: NÚMEROS PRIMOS, COMPUESTOS, MÚLTIPLOS Y DIVISORES

Introducción

En matemáticas y programación, los **números primos, compuestos, múltiplos y divisores** son conceptos fundamentales.
Python nos permite trabajar con ellos fácilmente mediante operaciones aritméticas, condicionales y bucles.

En esta lección aprenderás a **detectar números primos**, **distinguirlos de los compuestos**, **calcular múltiplos** y **encontrar divisores** de un número usando Python.

Números primos y compuestos

Un **número primo** es aquel que **solo tiene dos divisores positivos: 1 y él mismo**.
Ejemplos: 2, 3, 5, 7, 11, 13, 17, 19.

Un **número compuesto** tiene **más de dos divisores**.
Ejemplos: 4, 6, 8, 9, 10, 12, 15.

Verificar si un número es primo

El método más común consiste en comprobar si el número es divisible por algún valor entre **2 y la raíz cuadrada del número**.
Si no es divisible por ninguno, es **primo**.

Ejemplo:

```
n = int(input("Introduce un número: "))
es_primo = True
if n <= 1:
    es_primo = False
else:
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            es_primo = False
            break
if es_primo:
    print("Es primo")
else:
    print("Es compuesto")
```

Explicación:

- Si el número es menor o igual a 1, **no es primo**.

- Si ningún número entre 2 y √n lo divide exactamente, **sí es primo**.

Función para determinar si un número es primo

Podemos definir una función para reutilizar el código:

```
def es_primo(n):
    if n <= 1:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True


print(es_primo(7))
print(es_primo(10))
```

Resultado:
True
False

Listar los números primos hasta un límite

Podemos generar todos los números primos en un rango:

```
limite = int(input("Introduce el límite: "))
for n in range(2, limite + 1):
    es_primo = True
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            es_primo = False
            break
    if es_primo:
        print(n)
```

Números compuestos

Un **número compuesto** es aquel que **tiene más de dos divisores**.
Podemos comprobarlo usando la función anterior:

```
n = int(input("Número: "))
if es_primo(n):
    print("Es primo")
```

```
else:
    print("Es compuesto")
```

Múltiplos

Un número **A es múltiplo de B** si al dividir A entre B, el resto es cero (**A % B == 0**).
Ejemplo:

```
a = 12
b = 3
if a % b == 0:
    print(a, "es múltiplo de", b)
else:
    print(a, "no es múltiplo de", b)
```

Resultado:
12 es múltiplo de 3

Calcular múltiplos de un número

Podemos mostrar los primeros múltiplos de un número con un bucle for:

```
n = int(input("Número: "))
for i in range(1, 11):
    print(n * i)
```

Resultado (si n = 4):
4
8
12
16
20
24
28
32
36
40

Divisores

Un **divisor** de un número N es todo número que **divide a N sin dejar resto**.
Para hallarlos:

```
n = int(input("Número: "))
for i in range(1, n + 1):
    if n % i == 0:
```

```
        print(i)
```

Ejemplo: si n = 12, salida:
1 2 3 4 6 12

Contar el número de divisores

```
n = int(input("Número: "))
contador = 0
for i in range(1, n + 1):
    if n % i == 0:
        contador += 1
print("Tiene", contador, "divisores")
```

Clasificación usando divisores

Si un número tiene exactamente **2 divisores**, es **primo**.
Si tiene **más de 2**, es **compuesto**.

```
n = int(input("Número: "))
divisores = 0
for i in range(1, n + 1):
    if n % i == 0:
        divisores += 1
if divisores == 2:
    print("Es primo")
elif divisores > 2:
    print("Es compuesto")
else:
    print("No es primo ni compuesto")
```

Optimización: probar solo hasta la raíz cuadrada

Cuando se buscan divisores o se verifica si un número es primo, basta revisar hasta **la raíz cuadrada de n**, lo que mejora el rendimiento.
Esto es importante en algoritmos que manejan números grandes.

Uso de listas por comprensión

Podemos generar divisores usando una sola línea:

```
n = 28
divisores = [i for i in range(1, n + 1) if n % i == 0]
```

```
print(divisores)
```

Resultado:
[1, 2, 4, 7, 14, 28]

Resumen general

- **Número primo:** solo tiene dos divisores (1 y él mismo).
- **Número compuesto:** tiene más de dos divisores.
- **Múltiplo:** A es múltiplo de B si A % B == 0.
- **Divisor:** D es divisor de N si N % D == 0.
- **Las potencias y raíces no determinan primalidad.**
- **El uso de range() y bucles for permite hallar divisores y múltiplos fácilmente.**

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE NÚMEROS PRIMOS Y COMPUESTOS, MÚLTIPLOS Y DIVISORES

1. Comprobar si un número es primo

```
n = int(input("Número: "))
es_primo = True
for i in range(2, n):
    if n % i == 0:
        es_primo = False
        break
print("Primo" if es_primo else "Compuesto")
```

2. Mostrar todos los números primos del 1 al 50

```
for n in range(2, 51):
    for i in range(2, n):
        if n % i == 0:
            break
    else:
        print(n)
```

3. Contar cuántos primos hay hasta un número dado

```
limite = int(input("Hasta: "))
contador = 0
for n in range(2, limite + 1):
    for i in range(2, n):
        if n % i == 0:
```

```
            break
    else:
        contador += 1
print("Cantidad de primos:", contador)
```

4. Verificar si un número es compuesto

```
n = int(input("Número: "))
if n > 1:
    for i in range(2, n):
        if n % i == 0:
            print("Compuesto")
            break
    else:
        print("Primo")
else:
    print("No válido")
```

5. Calcular los divisores de un número

```
n = int(input("Número: "))
for i in range(1, n + 1):
    if n % i == 0:
        print(i)
```

6. Contar los divisores de un número

```
n = int(input("Número: "))
contador = 0
for i in range(1, n + 1):
    if n % i == 0:
        contador += 1
print("Tiene", contador, "divisores")
```

7. Listar los múltiplos de un número

```
n = int(input("Número: "))
for i in range(1, 11):
    print(n * i)
```

8. Verificar si un número es múltiplo de otro

```
a = int(input("A: "))
b = int(input("B: "))
print(a, "es múltiplo de", b) if a % b == 0 else print("No es múltiplo")
```

9. Mostrar los divisores comunes de dos números

```
a = int(input("A: "))
b = int(input("B: "))
for i in range(1, min(a, b) + 1):
    if a % i == 0 and b % i == 0:
        print(i)
```

10. Calcular el máximo divisor común (MCD)

```
a = int(input("A: "))
b = int(input("B: "))
while b != 0:
    a, b = b, a % b
print("MCD:", a)
```

11. Calcular el mínimo común múltiplo (MCM)

```
a = int(input("A: "))
b = int(input("B: "))
def mcd(a,b):
    while b != 0:
        a,b = b,a%b
    return a
print("MCM:", abs(a*b)//mcd(a,b))
```

12. Mostrar los divisores pares de un número

```
n = int(input("Número: "))
for i in range(1, n + 1):
    if n % i == 0 and i % 2 == 0:
        print(i)
```

13. Mostrar los divisores impares de un número

```
n = int(input("Número: "))
for i in range(1, n + 1):
    if n % i == 0 and i % 2 != 0:
        print(i)
```

14. Contar cuántos divisores tiene un número primo

```
n = int(input("Número: "))
contador = 0
for i in range(1, n + 1):
    if n % i == 0:
```

```
        contador += 1
print("Divisores:", contador)
```

15.Generar lista de divisores con comprensión

```
n = 24
divisores = [i for i in range(1, n + 1) if n % i == 0]
print(divisores)
```

16.Mostrar los múltiplos de 3 entre 1 y 100

```
for i in range(3, 101, 3):
    print(i)
```

17.Mostrar los múltiplos comunes de 4 y 6 hasta 50

```
for i in range(1, 51):
    if i % 4 == 0 and i % 6 == 0:
        print(i)
```

18.Verificar si un número tiene más de 4 divisores

```
n = int(input("Número: "))
contador = 0
for i in range(1, n + 1):
    if n % i == 0:
        contador += 1
print("Tiene más de 4 divisores" if contador > 4 else "No tiene más de 4 divisores")
```

19.Mostrar los primeros 10 números compuestos

```
contador = 0
n = 4
while contador < 10:
    for i in range(2, n):
        if n % i == 0:
            print(n)
            contador += 1
            break
    n += 1
```

20.Comprobar si un número es primo y mostrar sus divisores

```
n = int(input("Número: "))
divisores = [i for i in range(1, n + 1) if n % i == 0]
print("Divisores:", divisores)
if len(divisores) == 2:
```

```
    print("Es primo")
else:
    print("Es compuesto")
```

---

8. LECCIÓN: MÁXIMO COMÚN DIVISOR (MCD)

INTRODUCCIÓN

El **Máximo Común Divisor (MCD)** de dos o más números enteros es el **mayor número entero que divide exactamente a todos ellos**.
Conocer el MCD es fundamental en matemáticas para **simplificar fracciones, resolver problemas de divisibilidad y encontrar múltiplos comunes**.

En Python podemos calcular el MCD usando **algoritmos clásicos** como **el método de Euclides**, o con la función incorporada `math.gcd()`.

---

1. USO DE math.gcd()

Python incluye la función **math.gcd(a, b)** para calcular directamente el MCD de dos números.
Ejemplo:

```
import math


a = 24
b = 36
mcd = math.gcd(a, b)
print("MCD:", mcd)
```

Resultado: 12

Explicación:

- `math.gcd(a, b)` devuelve el **mayor número que divide exactamente a a y b**.
- Funciona solo para **enteros positivos o cero**.

---

2. MÉTODO DE EUCLIDES

El **algoritmo de Euclides** es un método eficiente para calcular el MCD:

1. Tomar dos números a y b.
2. Mientras b ≠ 0:
    - asignar a, b = b, a % b
3. El MCD es a cuando b = 0

Ejemplo:

```
a = 48
b = 18
while b != 0:
    a, b = b, a % b
```

```
print("MCD:", a)
```

Resultado: 6

Explicación paso a paso:

- 48 % 18 = 12 → a=18, b=12
- 18 % 12 = 6 → a=12, b=6
- 12 % 6 = 0 → a=6, b=0 → MCD=6

---

3. MCD DE MÁS DE DOS NÚMEROS

Podemos usar `math.gcd()` de forma iterativa:

```
import math

numeros = [24, 36, 60]
mcd = numeros[0]
for n in numeros[1:]:
    mcd = math.gcd(mcd, n)
print("MCD:", mcd)
```

Resultado: 12

Explicación:

- Se calcula el MCD de los dos primeros números, luego el resultado se combina con el siguiente.

---

4. MCD CON FUNCIÓN PERSONALIZADA

Podemos crear una función para reutilizar el cálculo:

```
def mcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

print(mcd(48, 18))
```

---

5. MCD Y PROPIEDADES DE DIVISIBILIDAD

- Si **a % MCD == 0** y **b % MCD == 0**, entonces MCD divide a ambos.
- **MCD(a, b) * MCM(a, b) = a * b** (propiedad importante).

Ejemplo:

```
import math
```

```
a = 12
b = 18
mcd = math.gcd(a, b)
mcm = abs(a*b)//mcd
print("MCD:", mcd, "MCM:", mcm)
```

Resultado: MCD: 6, MCM: 36

---

6. MCD CON RECURSIÓN

También se puede implementar el algoritmo de Euclides usando **recursión**:

```
def mcd(a, b):
    if b == 0:
        return a
    else:
        return mcd(b, a % b)

print(mcd(48, 18))
```

---

7. MCD Y FRACCIONES

El MCD se usa para **simplificar fracciones**:

```
numerador = 36
denominador = 60
import math
m = math.gcd(numerador, denominador)
print("Fracción simplificada:", numerador//m, "/", denominador//m)
```

Resultado: 3 / 5

---

8. MCD DE LISTA DE NÚMEROS

Podemos usar **functools.reduce** para calcular el MCD de una lista grande:

```
from functools import reduce
import math

numeros = [48, 60, 72]
mcd_total = reduce(math.gcd, numeros)
print("MCD de la lista:", mcd_total)
```

Resultado: 12

---

9. EJEMPLOS PRÁCTICOS

- Determinar si dos números son coprimos: si **MCD = 1** → coprimos.
- Simplificar fracciones.
- Resolver problemas de reparto exacto.
- Encontrar divisores comunes máximos.

---

10. RESUMEN GENERAL

- **MCD**: el mayor número que divide exactamente a dos o más enteros.
- **math.gcd(a, b)**: función incorporada para calcularlo.
- **Algoritmo de Euclides**: método clásico y eficiente.
- Se puede extender a **más de dos números**.
- Se aplica en fracciones, divisibilidad y coprimos.

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE MÁXIMO COMÚN DIVISOR

1. MCD de dos números usando math.gcd()

```
import math
print(math.gcd(48, 18))
```

2. MCD de dos números usando Euclides iterativo

```
a, b = 48, 18
while b != 0:
    a, b = b, a % b
print(a)
```

3. MCD usando función personalizada

```
def mcd(a,b):
    while b != 0:
        a,b = b, a%b
    return a
print(mcd(56, 98))
```

4. MCD de tres números

```
import math
nums = [24, 36, 60]
```

```
m = nums[0]
for n in nums[1:]:
    m = math.gcd(m, n)
print(m)
```

5. MCD de lista grande usando reduce

```
from functools import reduce
import math
numeros = [48, 60, 72, 84]
mcd_total = reduce(math.gcd, numeros)
print(mcd_total)
```

6. Verificar si dos números son coprimos

```
import math
a, b = 35, 64
print("Coprimos" if math.gcd(a,b)==1 else "No coprimos")
```

7. Simplificar fracción usando MCD

```
import math
num, den = 36, 60
m = math.gcd(num, den)
print(num//m, "/", den//m)
```

8. MCD usando recursión

```
def mcd(a,b):
    if b==0:
        return a
    return mcd(b, a%b)
print(mcd(270,192))
```

9. MCD y MCM combinados

```
import math
a,b = 12, 18
mcd = math.gcd(a,b)
mcm = abs(a*b)//mcd
print("MCD:", mcd, "MCM:", mcm)
```

10. Calcular MCD de números ingresados por usuario

```
import math
a = int(input("A: "))
b = int(input("B: "))
```

```
print("MCD:", math.gcd(a,b))
```

11.MCD de cuatro números

```
from functools import reduce
import math
numeros = [48, 64, 80, 96]
print(reduce(math.gcd, numeros))
```

12.MCD de dos números con input y recursión

```
def mcd(a,b):
    return a if b==0 else mcd(b,a%b)
a = int(input("A: "))
b = int(input("B: "))
print(mcd(a,b))
```

13.Verificar si MCD divide a ambos números

```
import math
a,b = 42, 56
m = math.gcd(a,b)
print(a % m == 0 and b % m == 0)
```

14.Encontrar MCD de números pares

```
import math
a,b = 48, 64
print(math.gcd(a,b))
```

15.Encontrar MCD de números impares

```
import math
a,b = 35, 105
print(math.gcd(a,b))
```

16.MCD de números grandes

```
import math
print(math.gcd(123456, 789012))
```

17.MCD de números negativos

```
import math
print(math.gcd(-48, 18))
```

18.Verificar si MCD es 1 para coprimos

```
import math
a,b = 17, 31
print("Coprimos" if math.gcd(a,b)==1 else "No coprimos")
```

19.MCD en lista con función personalizada

```
def mcd(a,b):
    while b!=0:
        a,b = b,a%b
    return a
nums = [18, 24, 30]
m = nums[0]
for n in nums[1:]:
    m = mcd(m,n)
print(m)
```

20.MCD y simplificación de fracción con input

```
import math
num = int(input("Numerador: "))
den = int(input("Denominador: "))
m = math.gcd(num,den)
print("Fracción simplificada:", num//m,"/",den//m)
```

---

9. LECCIÓN: MÍNIMO COMÚN MÚLTIPLO (MCM)

INTRODUCCIÓN

El **Mínimo Común Múltiplo (MCM)** de dos o más números enteros es el **menor número positivo que es múltiplo de todos ellos**.
El MCM es útil en **matemáticas**, especialmente en la **suma y resta de fracciones, planificación de ciclos y sincronización de eventos**.

En Python podemos calcularlo usando **fórmulas matemáticas**, la función **math.gcd()** o mediante **algoritmos iterativos**.

---

1. RELACIÓN ENTRE MCD Y MCM

Existe una propiedad fundamental:

```
MCM(a, b) * MCD(a, b) = a * b
```

Por lo tanto, si conocemos el **MCD**, podemos calcular el **MCM**:

```
MCM(a, b) = abs(a * b) // MCD(a, b)
```

Ejemplo:

```
import math

a = 12

b = 18

mcd = math.gcd(a, b)

mcm = abs(a*b)//mcd

print("MCM:", mcm)
```

Resultado: 36

---

2. CÁLCULO DEL MCM DE DOS NÚMEROS

Método usando bucle:

```
a = 12

b = 18

mayor = max(a, b)

while True:

    if mayor % a == 0 and mayor % b == 0:

        mcm = mayor

        break

    mayor += 1

print("MCM:", mcm)
```

Explicación:

- Comenzamos con el mayor de los dos números.
- Incrementamos hasta encontrar un número divisible por ambos.

---

3. MCM DE MÁS DE DOS NÚMEROS

Podemos usar la **propiedad asociativa** del MCM:

```
MCM(a, b, c) = MCM(MCM(a, b), c)
```

Ejemplo:

```
import math

def mcm(a, b):
    return abs(a*b) // math.gcd(a, b)

numeros = [4, 6, 8]
resultado = numeros[0]
for n in numeros[1:]:
    resultado = mcm(resultado, n)
print("MCM:", resultado)
```

Resultado: 24

---

4. FUNCIÓN PERSONALIZADA DE MCM

```
import math

def mcm(a, b):
    return abs(a*b) // math.gcd(a,b)

print(mcm(12,18))
```

---

5. MCM Y FRACCIONES

El MCM se usa para **encontrar denominadores comunes** al sumar o restar fracciones:

```
import math

a, b = 1, 3   # fracción 1/3
c, d = 1, 4   # fracción 1/4

mcm_den = abs(b*d) // math.gcd(b,d)
```

```
resultado = (a*(mcm_den//b) + c*(mcm_den//d))
print("Suma de fracciones:", resultado, "/", mcm_den)
```

Resultado: 7 / 12

---

6. MCM USANDO REDUCE PARA LISTAS

Podemos calcular el MCM de una lista de números usando **functools.reduce**:

```
from functools import reduce
import math

def mcm(a,b):
    return abs(a*b)//math.gcd(a,b)

numeros = [4, 6, 8, 12]
mcm_total = reduce(mcm, numeros)
print("MCM de la lista:", mcm_total)
```

---

7. EJEMPLOS PRÁCTICOS

- Sincronización de eventos periódicos.
- Resolución de fracciones con denominadores diferentes.
- Problemas de reparto equitativo con ciclos.
- Encontrar intervalos comunes de tiempo o distancia.

---

8. RESUMEN GENERAL

- **MCM**: menor número divisible por todos los números dados.
- **Propiedad clave:** MCM(a,b) = (a*b)//MCD(a,b)
- Se puede calcular con **bucles, funciones o reduce**.
- Se aplica en fracciones, ciclos y divisibilidad.

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE MÍNIMO COMÚN MÚLTIPLO

1. MCM de dos números usando math.gcd

```
import math
a, b = 12, 18
mcm = abs(a*b)//math.gcd(a,b)
print(mcm)
```

2. MCM de dos números usando bucle

```
a, b = 12, 18
mayor = max(a,b)
while True:
    if mayor % a == 0 and mayor % b == 0:
        print(mayor)
        break
    mayor += 1
```

3. MCM de tres números usando reduce

```
from functools import reduce
import math
def mcm(a,b):
    return abs(a*b)//math.gcd(a,b)
numeros = [4,6,8]
print(reduce(mcm,numeros))
```

4. Función personalizada de MCM

```
import math
def mcm(a,b):
    return abs(a*b)//math.gcd(a,b)
print(mcm(15,20))
```

5. MCM para lista de cuatro números

```
from functools import reduce
import math
def mcm(a,b):
    return abs(a*b)//math.gcd(a,b)
print(reduce(mcm,[5,10,15,20]))
```

6. MCM y simplificación de fracciones

```
import math
a,b = 1,3
c,d = 1,4
mcm_den = abs(b*d)//math.gcd(b,d)
resultado = a*(mcm_den//b) + c*(mcm_den//d)
print(resultado,'/',mcm_den)
```

7. MCM usando while con números iguales

```
a = 7
```

```
b = 7
mayor = max(a,b)
while True:
    if mayor % a == 0 and mayor % b == 0:
        print(mayor)
        break
    mayor += 1
```

8. MCM de números primos

```
import math
print(abs(3*5)//math.gcd(3,5))
```

9. MCM de números coprimos

```
import math
a,b = 8,15
print(abs(a*b)//math.gcd(a,b))
```

10. MCM de dos números con input

```
import math
a=int(input())
b=int(input())
print(abs(a*b)//math.gcd(a,b))
```

11. MCM de tres números ingresados por usuario

```
from functools import reduce
import math
def mcm(a,b):
    return abs(a*b)//math.gcd(a,b)
nums=[int(input()) for _ in range(3)]
print(reduce(mcm,nums))
```

12. MCM de lista de números grandes

```
from functools import reduce
import math
def mcm(a,b):
    return abs(a*b)//math.gcd(a,b)
nums=[120,180,240]
print(reduce(mcm,nums))
```

13. Verificar si MCM es divisible por cada número

```
from functools import reduce
```

```
import math
def mcm(a,b):
    return abs(a*b)//math.gcd(a,b)
nums=[4,6,8]
m=reduce(mcm,nums)
print(all(m%n==0 for n in nums))
```

14.MCM de dos números negativos

```
import math
a,b=-12,18
print(abs(a*b)//math.gcd(a,b))
```

15.MCM de dos números iguales

```
import math
a,b=14,14
print(abs(a*b)//math.gcd(a,b))
```

16.MCM y MCD juntos

```
import math
a,b=12,18
mcd=math.gcd(a,b)
mcm=abs(a*b)//mcd
print("MCD:",mcd,"MCM:",mcm)
```

17.MCM de lista con función recursiva

```
import math
from functools import reduce
def mcm(a,b):
    return abs(a*b)//math.gcd(a,b)
nums=[5,10,15]
print(reduce(mcm,nums))
```

18.MCM de tres números consecutivos

```
import math
def mcm(a,b):
    return abs(a*b)//math.gcd(a,b)
print(reduce(mcm,[6,7,8]))
```

19.MCM de números pares

```
import math
def mcm(a,b):
```

```
    return abs(a*b)//math.gcd(a,b)
print(reduce(mcm,[2,4,6,8]))
```

20.MCM aplicado a denominadores de fracciones

```
import math
from functools import reduce
def mcm(a,b):
    return abs(a*b)//math.gcd(a,b)
den=[3,4,5]
m=reduce(mcm,den)
print("Denominador común:",m)
```

---

10. LECCIÓN: PRINCIPIOS FUNDAMENTALES DE LA DIVISIBILIDAD

En esta lección estudiaremos los **principios fundamentales de la divisibilidad**, cómo aplicarlos en **Python**, y cómo resolver **problemas prácticos** con ejemplos de código.

---

1. CONCEPTO DE DIVISIBILIDAD

Un número **a** es **divisible** por otro número **b** si al dividir **a / b**, el **resto** es **cero**.

Ejemplo:
10 es divisible por 5 porque 10 ÷ 5 = 2 y el resto es 0.
10 no es divisible por 3 porque 10 ÷ 3 = 3 y el resto es 1.

En **Python**, se usa el **operador módulo (%)** para comprobar la divisibilidad:

Ejemplo:
```
a = 10
b = 5
if a % b == 0:
print("a es divisible por b")
else:
print("a no es divisible por b")
```

Explicación:
El operador **%** devuelve el **resto** de la división.
Si el resto es **0**, significa que el número es divisible.

---

2. DIVISORES Y MÚLTIPLOS

- **Divisor:** un número que divide exactamente a otro.
  Ejemplo: los divisores de 12 son 1, 2, 3, 4, 6, 12.
- **Múltiplo:** un número que resulta de multiplicar otro por un entero.
  Ejemplo: los múltiplos de 3 son 3, 6, 9, 12, 15, etc.

Ejemplo en Python para obtener divisores:
```
n = 12
for i in range(1, n + 1):
if n % i == 0:
print(i, "es divisor de", n)
```

Ejemplo para obtener múltiplos:
```
num = 3
for i in range(1, 6):
print(num * i)
```

---

3. REGLAS DE DIVISIBILIDAD

En matemáticas existen reglas que permiten saber si un número es divisible por otro sin hacer la división.
Podemos aplicarlas en **Python** con condiciones.

Divisible por 2: el número termina en cifra par.
Divisible por 3: la suma de sus dígitos es múltiplo de 3.
Divisible por 5: termina en 0 o 5.
Divisible por 10: termina en 0.

Ejemplo:

```
n = 45
if n % 2 == 0:
print("Divisible por 2")
if sum(int(d) for d in str(n)) % 3 == 0:
print("Divisible por 3")
if n % 5 == 0:
print("Divisible por 5")
```

---

4. DIVISORES COMUNES Y MÚLTIPLOS COMUNES

Dos números pueden tener divisores comunes (por ejemplo, 2 divide a 6 y 8).
También pueden tener múltiplos comunes (por ejemplo, 12 es múltiplo común de 3 y 4).

Podemos calcular:

- **MCD (Máximo Común Divisor)** con la función **math.gcd()**.
- **MCM (Mínimo Común Múltiplo)** con una fórmula matemática.

Ejemplo:

```
import math

a = 12
b = 18
print("MCD:", math.gcd(a, b))
mcm = abs(a * b) // math.gcd(a, b)
print("MCM:", mcm)
```

---

5. NÚMEROS COPRIMOS

Dos números son **coprimos** si su MCD es 1.

Ejemplo:

```
import math

a = 8
b = 15
if math.gcd(a, b) == 1:
```

```
print("Son coprimos")
else:
print("No son coprimos")
```

---

6. COMPROBACIÓN DE DIVISIBILIDAD EN LISTAS

Podemos verificar qué números de una lista son divisibles entre otros.

Ejemplo:

```
numeros = [10, 15, 21, 30, 42]
for n in numeros:
if n % 3 == 0:
print(n, "es divisible por 3")
```

---

7. FUNCIÓN GENERAL DE DIVISIBILIDAD

Podemos crear una **función personalizada** para reutilizar el código.

Ejemplo:

```
def es_divisible(a, b):
return a % b == 0

print(es_divisible(12, 3))
print(es_divisible(10, 4))
```

---

8. RELACIÓN ENTRE DIVISIBILIDAD Y PRIMOS

Un número **primo** solo tiene dos divisores: 1 y él mismo.
Un número **compuesto** tiene más de dos divisores.

Ejemplo:

```
n = 11
cont = 0
for i in range(1, n + 1):
if n % i == 0:
cont += 1
if cont == 2:
print("Es primo")
else:
print("Es compuesto")
```

---

9. SUMA DE DIVISORES Y PROPIEDADES

Podemos sumar los divisores de un número para estudiar propiedades como los **números perfectos**.

Ejemplo:

```
n = 6
suma = 0
for i in range(1, n):
if n % i == 0:
suma += i
if suma == n:
print("Es número perfecto")
```

---

10. APLICACIONES PRÁCTICAS DE LA DIVISIBILIDAD

Los principios de divisibilidad se usan en:

- Criptografía
- Simplificación de fracciones
- Algoritmos de búsqueda de patrones numéricos
- Programas de validación de números

PROGRAMAS RESUELTOS DE PRINCIPIOS FUNDAMENTALES DE LA DIVISIBILIDAD

1. Comprobar si un número es divisible por 7

   ```
   n = 49
   print(n % 7 == 0)
   ```

2. Mostrar todos los divisores de un número

   ```
   n = 20
   for i in range(1, n + 1):
   if n % i == 0:
   print(i)
   ```

3. Mostrar los múltiplos de un número hasta 100

   ```
   n = 8
   for i in range(1, 13):
   print(n * i)
   ```

4. Saber si un número es par o impar usando %

   ```
   n = 27
   print("Par" if n % 2 == 0 else "Impar")
   ```

5. Verificar si un número es divisible por 3 y 5

   ```
   n = 60
   print(n % 3 == 0 and n % 5 == 0)
   ```

6. Calcular el MCD de dos números
```
import math
print(math.gcd(24, 36))
```
7. Calcular el MCM de dos números
```
import math
a, b = 15, 20
mcm = abs(a * b) // math.gcd(a, b)
print(mcm)
```
8. Verificar si dos números son coprimos
```
import math
a, b = 14, 25
print(math.gcd(a, b) == 1)
```
9. Mostrar los números divisibles por 4 en una lista
```
lista = [8, 12, 15, 20, 33]
for n in lista:
if n % 4 == 0:
print(n)
```
10. Contar cuántos divisores tiene un número
```
n = 18
cont = 0
for i in range(1, n + 1):
if n % i == 0:
cont += 1
print(cont)
```
11. Mostrar los divisores comunes de dos números
```
a, b = 20, 30
for i in range(1, min(a, b) + 1):
if a % i == 0 and b % i == 0:
print(i)
```
12. Sumar todos los divisores de un número
```
n = 12
suma = sum(i for i in range(1, n + 1) if n % i == 0)
print(suma)
```
13. Determinar si un número es perfecto
```
n = 28
suma = sum(i for i in range(1, n) if n % i == 0)
print(suma == n)
```
14. Mostrar los múltiplos comunes de dos números
```
a, b = 3, 5
for i in range(1, 21):
if i % a == 0 and i % b == 0:
print(i)
```

15. Verificar si un número es divisible por 11

```
n = 121
print(n % 11 == 0)
```

16. Verificar si un número es divisible por 9

```
n = 81
print(sum(int(d) for d in str(n)) % 9 == 0)
```

17. Mostrar divisores pares de un número

```
n = 24
for i in range(2, n + 1, 2):
if n % i == 0:
print(i)
```

18. Mostrar divisores impares de un número

```
n = 45
for i in range(1, n + 1, 2):
if n % i == 0:
print(i)
```

19. Encontrar el menor divisor distinto de 1

```
n = 91
for i in range(2, n + 1):
if n % i == 0:
print(i)
break
```

20. Verificar si un número es múltiplo de otro

```
a, b = 45, 9
print(a % b == 0)
```

---

11. LECCIÓN: NÚMEROS FRACCIONARIOS Y SUS PROPIEDADES

INTRODUCCIÓN

Los **números fraccionarios** son números que representan una **parte de un todo**. Se escriben en la forma:

numerador / denominador

- **Numerador**: indica cuántas partes se toman.
- **Denominador**: indica en cuántas partes se divide el todo.

En Python, los números fraccionarios se pueden manejar usando:

- **Operaciones con flotantes (float)**
- **El módulo fractions.Fraction** para mantener precisión exacta

---

1. CREAR FRACCIONES CON fractions.Fraction

```
from fractions import Fraction


f1 = Fraction(3, 4)

f2 = Fraction(2, 5)

print(f1)

print(f2)
```

Resultado:
3/4
2/5

Explicación:

- `Fraction(a,b)` crea la fracción a/b
- Mantiene la precisión exacta, a diferencia de los floats.

---

2. OPERACIONES BÁSICAS

Suma

```
from fractions import Fraction


f1 = Fraction(3, 4)

f2 = Fraction(2, 5)

suma = f1 + f2

print(suma)
```

Resultado: 23/20

Resta

```
resta = f1 - f2
print(resta)
```

Resultado: 7/20

Multiplicación

```
producto = f1 * f2
print(producto)
```

Resultado: 6/20 → se simplifica a 3/10

División

```
division = f1 / f2
print(division)
```

Resultado: 15/8

---

3. SIMPLIFICACIÓN AUTOMÁTICA

El módulo `fractions` simplifica automáticamente:

```
Fraction(6,20)
```

Resultado: 3/10

---

4. CONVERSIÓN ENTRE FRACCIONES Y FLOAT

```
f = Fraction(3,4)
print(float(f))
```

Resultado: 0.75

También se puede crear una fracción a partir de un float:

```
Fraction(0.75)
```

Resultado: 3/4

---

5. PROPIEDADES GENERALES DE LAS FRACCIONES

1. **Propiedad conmutativa de la suma**: a/b + c/d = c/d + a/b
2. **Propiedad asociativa de la suma**: (a/b + c/d) + e/f = a/b + (c/d + e/f)
3. **Propiedad conmutativa de la multiplicación**: a/b * c/d = c/d * a/b
4. **Propiedad distributiva**: a/b*(c/d + e/f) = a/b*c/d* + *a/b*e/f
5. **Identidad de la suma**: a/b + 0 = a/b
6. **Identidad de la multiplicación**: a/b * 1 = a/b

---

6. OPERACIONES CON LISTAS DE FRACCIONES

Se puede sumar, multiplicar o simplificar varias fracciones usando **bucles**:

```
from fractions import Fraction

fracciones = [Fraction(1,2), Fraction(1,3), Fraction(1,4)]
suma_total = sum(fracciones)
print(suma_total)
```

Resultado: 13/12

---

7. COMPARACIÓN DE FRACCIONES

```
f1 = Fraction(3,4)
f2 = Fraction(2,3)
print(f1 > f2)   # True
print(f1 == f2)  # False
```

---

8. OTRAS FUNCIONES ÚTILES

- `f.numerator` → devuelve numerador
- `f.denominator` → devuelve denominador
- `f.limit_denominator(max_den)` → aproxima fracción a denominador máximo

Ejemplo:

```
f = Fraction(22,7)
print(f.numerator)
print(f.denominator)
print(f.limit_denominator(10))  # Aproxima a fracción con denominador ≤ 10
```

---

9. FRACCIONES Y DECIMALES

Para trabajar con decimales y fracciones juntas:

```
from fractions import Fraction

f = Fraction(1,3)
d = 0.25
resultado = f + Fraction(d)
print(resultado)
```

Resultado: 7/12

---

10. RESUMEN GENERAL

- Las **fracciones** representan partes de un todo.
- El módulo **fractions** permite manipulación precisa.
- Operaciones principales: **suma, resta, multiplicación, división**.
- Propiedades: **conmutativa, asociativa, distributiva, identidad**.
- Se pueden comparar, simplificar y convertir a float.

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE NÚMEROS FRACCIONARIOS

1. Crear fracción y mostrarla

```
from fractions import Fraction
f = Fraction(3,4)
print(f)
```

2. Sumar dos fracciones

```
from fractions import Fraction
f1 = Fraction(3,4)
f2 = Fraction(2,5)
print(f1 + f2)
```

3. Restar dos fracciones

```
from fractions import Fraction
f1 = Fraction(3,4)
f2 = Fraction(2,5)
print(f1 - f2)
```

4. Multiplicar dos fracciones

```
from fractions import Fraction
f1 = Fraction(3,4)
f2 = Fraction(2,5)
print(f1 * f2)
```

5. Dividir dos fracciones

```
from fractions import Fraction
f1 = Fraction(3,4)
f2 = Fraction(2,5)
print(f1 / f2)
```

6. Sumar lista de fracciones

```
from fractions import Fraction
```

```
fracciones = [Fraction(1,2), Fraction(1,3), Fraction(1,4)]
print(sum(fracciones))
```

7. Comparar fracciones

```
from fractions import Fraction
f1 = Fraction(3,4)
f2 = Fraction(2,3)
print(f1 > f2)
```

8. Fracción a decimal

```
from fractions import Fraction
f = Fraction(3,4)
print(float(f))
```

9. Crear fracción a partir de decimal

```
from fractions import Fraction
f = Fraction(0.75)
print(f)
```

10. Fracción simplificada automáticamente

```
from fractions import Fraction
f = Fraction(6,20)
print(f)
```

11. Numerador y denominador

```
from fractions import Fraction
f = Fraction(3,4)
print(f.numerator)
print(f.denominator)
```

12. Aproximar fracción con denominador máximo

```
from fractions import Fraction
f = Fraction(22,7)
print(f.limit_denominator(10))
```

13. Propiedad conmutativa suma

```
from fractions import Fraction
f1 = Fraction(1,2)
f2 = Fraction(1,3)
print(f1 + f2 == f2 + f1)
```

14. Propiedad asociativa suma

```
from fractions import Fraction
```

```
f1 = Fraction(1,2)
f2 = Fraction(1,3)
f3 = Fraction(1,4)
print((f1+f2)+f3 == f1+(f2+f3))
```

15.Propiedad conmutativa multiplicación

```
from fractions import Fraction
f1 = Fraction(2,3)
f2 = Fraction(3,5)
print(f1*f2 == f2*f1)
```

16.Propiedad distributiva

```
from fractions import Fraction
f1 = Fraction(2,3)
f2 = Fraction(1,4)
f3 = Fraction(1,6)
print(f1*(f2+f3) == f1*f2 + f1*f3)
```

17.Identidad suma

```
from fractions import Fraction
f = Fraction(3,5)
print(f + 0 == f)
```

18.Identidad multiplicación

```
from fractions import Fraction
f = Fraction(3,5)
print(f * 1 == f)
```

19.Sumar fracción y decimal

```
from fractions import Fraction
f = Fraction(1,3)
d = 0.25
print(f + Fraction(d))
```

20.Multiplicar varias fracciones de lista

```
from fractions import Fraction
fracciones = [Fraction(1,2), Fraction(2,3), Fraction(3,4)]
producto = 1
for f in fracciones:
    producto *= f
print(producto)
```

12. LECCIÓN: REDUCCIÓN Y SIMPLIFICACIÓN DE QUEBRADOS

INTRODUCCIÓN

Los **quebrados** son números fraccionarios expresados como **numerador / denominador**.
La **reducción o simplificación** consiste en escribir un quebrado en su **forma más simple**, manteniendo el mismo valor.

Forma general:

```
a / b
```

- **a**: numerador
- **b**: denominador
- Se simplifica dividiendo ambos por su **MCD (Máximo Común Divisor)**

Python permite trabajar con quebrados y simplificarlos usando:

- El módulo **fractions.Fraction**
- Funciones con **math.gcd()**

---

1. CREAR QUEBRADOS CON fractions.Fraction

```
from fractions import Fraction


q = Fraction(8,12)
print(q)
```

Resultado: 2/3

Explicación:

- La fracción se **simplifica automáticamente** al crear el objeto Fraction.

---

2. SIMPLIFICACIÓN MANUAL USANDO MCD

```
import math


numerador = 8
denominador = 12
mcd = math.gcd(numerador, denominador)
simplificado = (numerador // mcd, denominador // mcd)
print(simplificado)
```

Resultado: (2,3)

Explicación:

- Se divide **numerador y denominador** por su MCD para simplificar.

---

3. SUMA DE QUEBRADOS Y SIMPLIFICACIÓN

```
from fractions import Fraction


q1 = Fraction(3,4)
q2 = Fraction(2,5)
suma = q1 + q2
print(suma)
```

Resultado: 23/20

Explicación:

- La suma se realiza automáticamente y la fracción se simplifica al final.

---

4. RESTA, MULTIPLICACIÓN Y DIVISIÓN DE QUEBRADOS

```
from fractions import Fraction


q1 = Fraction(3,4)
q2 = Fraction(2,5)

resta = q1 - q2
producto = q1 * q2
division = q1 / q2

print(resta, producto, division)
```

Resultado: 7/20 3/10 15/8

---

5. CONVERSIÓN ENTRE QUEBRADO Y FLOAT

```
from fractions import Fraction


q = Fraction(7,4)
print(float(q))
```

Resultado: 1.75

---

6. PROPÓSITOS DE SIMPLIFICACIÓN

- Facilita **operaciones con fracciones**

- Permite **comparar fracciones** fácilmente
- Es útil para **mostrar resultados más claros**

---

7. CREAR FUNCIONES DE SIMPLIFICACIÓN

```
import math

def simplificar(numerador, denominador):
    mcd = math.gcd(numerador, denominador)
    return (numerador // mcd, denominador // mcd)

print(simplificar(20,30))
```

Resultado: (2,3)

---

8. REDUCCIÓN DE LISTA DE QUEBRADOS

```
from fractions import Fraction

quebrados = [Fraction(4,8), Fraction(10,15), Fraction(18,24)]
simplificados = [q for q in quebrados]
print(simplificados)
```

Resultado: [1/2, 2/3, 3/4]

---

9. PROPIEDADES DE LOS QUEBRADOS SIMPLIFICADOS

- La fracción **mantiene su valor**
- Se cumplen todas las **propiedades de la aritmética**: conmutativa, asociativa, distributiva
- Las operaciones con fracciones simplificadas son **más sencillas y precisas**

---

10. RESUMEN GENERAL

- Un **quebrado** es un número fraccionario a/b
- La **simplificación** se logra dividiendo numerador y denominador por su MCD
- Python permite simplificar automáticamente con **fractions.Fraction** o manualmente con **math.gcd()**
- Simplificar facilita operaciones, comparación y presentación de resultados

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE REDUCCIÓN Y SIMPLIFICACIÓN DE QUEBRADOS

1. Crear un quebrado y simplificar automáticamente

```
from fractions import Fraction
q = Fraction(8,12)
print(q)
```

2. Simplificar manualmente usando math.gcd

```
import math
numerador, denominador = 8,12
mcd = math.gcd(numerador,denominador)
print(numerador//mcd, denominador//mcd)
```

3. Sumar quebrados y simplificar

```
from fractions import Fraction
q1 = Fraction(3,4)
q2 = Fraction(2,5)
print(q1+q2)
```

4. Restar quebrados y simplificar

```
from fractions import Fraction
q1 = Fraction(3,4)
q2 = Fraction(2,5)
print(q1-q2)
```

5. Multiplicar quebrados y simplificar

```
from fractions import Fraction
q1 = Fraction(3,4)
q2 = Fraction(2,5)
print(q1*q2)
```

6. Dividir quebrados y simplificar

```
from fractions import Fraction
q1 = Fraction(3,4)
q2 = Fraction(2,5)
print(q1/q2)
```

7. Convertir quebrado a decimal

```
from fractions import Fraction
q = Fraction(7,4)
print(float(q))
```

8. Crear función de simplificación

```
import math
def simplificar(n,d):
    mcd = math.gcd(n,d)
    return (n//mcd,d//mcd)
print(simplificar(20,30))
```

9. Simplificar lista de quebrados

```
from fractions import Fraction
quebrados=[Fraction(4,8),Fraction(10,15),Fraction(18,24)]
print([q for q in quebrados])
```

10. Comparar quebrados simplificados

```
from fractions import Fraction
q1=Fraction(2,3)
q2=Fraction(4,6)
print(q1==q2)
```

11. Sumar tres quebrados

```
from fractions import Fraction
q1=Fraction(1,2)
q2=Fraction(1,3)
q3=Fraction(1,4)
print(q1+q2+q3)
```

12. Multiplicar varios quebrados

```
from fractions import Fraction
quebrados=[Fraction(1,2),Fraction(2,3),Fraction(3,4)]
prod=1
for q in quebrados:
    prod*=q
print(prod)
```

13. Restar varios quebrados

```
from fractions import Fraction
q1=Fraction(5,6)
q2=Fraction(1,2)
q3=Fraction(1,3)
print(q1-q2-q3)
```

14. Simplificar quebrado de número negativo

```
from fractions import Fraction
q = Fraction(-8,12)
print(q)
```

15.Aproximar quebrado a decimal

```
from fractions import Fraction
q=Fraction(7,3)
print(float(q))
```

16.Sumar quebrado y decimal

```
from fractions import Fraction
q=Fraction(1,2)
d=0.25
print(q+Fraction(d))
```

17.Multiplicar quebrado por entero

```
from fractions import Fraction
q=Fraction(3,4)
print(q*5)
```

18.Dividir entero por quebrado

```
from fractions import Fraction
q=Fraction(3,4)
print(5/q)
```

19.Simplificar quebrado grande

```
from fractions import Fraction
q=Fraction(120,180)
print(q)
```

20.Sumar y simplificar fracciones con diferente denominador

```
from fractions import Fraction
q1=Fraction(5,6)
q2=Fraction(7,8)
print(q1+q2)
```

---

13. LECCIÓN: OPERACIONES CON NÚMEROS FRACCIONARIOS

INTRODUCCIÓN

Los **números fraccionarios** representan **una parte de un todo** y se escriben como **numerador / denominador**.
En Python, se pueden manejar con:

- **Flotantes (float)** para aproximaciones
- **fractions.Fraction** para precisión exacta

Trabajar con fracciones permite **sumar, restar, multiplicar y dividir** sin perder exactitud.

---

1. CREAR FRACCIONES

```
from fractions import Fraction


f1 = Fraction(3,4)
f2 = Fraction(2,5)
print(f1)
print(f2)
```

Resultado:
3/4
2/5

Explicación:

- `Fraction(a,b)` crea la fracción **a/b** y la simplifica automáticamente.

---

2. SUMA DE FRACCIONES

```
suma = f1 + f2
print(suma)
```

Resultado: 23/20

Explicación:

- Python calcula el **MCM de los denominadores** y realiza la suma exacta.

---

3. RESTA DE FRACCIONES

```
resta = f1 - f2
print(resta)
```

Resultado: 7/20

---

4. MULTIPLICACIÓN DE FRACCIONES

```
producto = f1 * f2
print(producto)
```

Resultado: 3/10

Explicación:

- Se multiplican **numeradores entre sí** y **denominadores entre sí**, y se simplifica automáticamente.

---

5. DIVISIÓN DE FRACCIONES

```
division = f1 / f2
print(division)
```

Resultado: 15/8

Explicación:

- La división de fracciones se realiza multiplicando la primera por el **recíproco** de la segunda.

---

6. CONVERSIÓN ENTRE FRACCIONES Y DECIMALES

```
print(float(f1))
```

Resultado: 0.75

```
f = Fraction(0.2)
print(f)
```

Resultado: 1/5

Explicación:

- Las fracciones pueden convertirse a decimal usando **float()**
- Los decimales se pueden convertir a fracciones exactas con **Fraction()**

---

7. SIMPLIFICACIÓN AUTOMÁTICA

```
f = Fraction(6,20)
print(f)
```

Resultado: 3/10

Explicación:

- fractions.Fraction simplifica automáticamente dividiendo numerador y denominador por su **MCD**

---

8. PROPIEDADES IMPORTANTES

- **Conmutativa:** f1 + f2 = f2 + f1 ; f1 * f2 = f2 * f1
- **Asociativa:** (f1 + f2) + f3 = f1 + (f2 + f3)
- **Distributiva:** f1*(f2 + f3) = f1*f2* + *f1*f3
- **Identidad de suma:** f + 0 = f
- **Identidad de multiplicación:** f * 1 = f

---

9. OPERACIONES CON LISTAS DE FRACCIONES

```
fracciones = [Fraction(1,2), Fraction(1,3), Fraction(1,4)]
suma_total = sum(fracciones)
print(suma_total)
```

Resultado: 13/12

```
producto_total = 1
for f in fracciones:
    producto_total *= f
print(producto_total)
```

Resultado: 1/24

---

10. RESUMEN GENERAL

- Los **números fraccionarios** representan partes de un todo.
- Se usan `Fraction` para mantener **precisión exacta**.
- Operaciones principales: **suma, resta, multiplicación, división**
- Las fracciones se **simplifican automáticamente**
- Propiedades importantes: **conmutativa, asociativa, distributiva, identidad**

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE OPERACIONES CON FRACCIONES

1. Crear fracción

```
from fractions import Fraction
f = Fraction(3,4)
print(f)
```

2. Sumar dos fracciones

```
from fractions import Fraction
f1 = Fraction(3,4)
```

```
f2 = Fraction(2,5)
print(f1+f2)
```

3. Restar dos fracciones

```
from fractions import Fraction
f1 = Fraction(3,4)
f2 = Fraction(2,5)
print(f1-f2)
```

4. Multiplicar dos fracciones

```
from fractions import Fraction
f1 = Fraction(3,4)
f2 = Fraction(2,5)
print(f1*f2)
```

5. Dividir dos fracciones

```
from fractions import Fraction
f1 = Fraction(3,4)
f2 = Fraction(2,5)
print(f1/f2)
```

6. Sumar lista de fracciones

```
from fractions import Fraction
fracciones = [Fraction(1,2), Fraction(1,3), Fraction(1,4)]
print(sum(fracciones))
```

7. Multiplicar lista de fracciones

```
from fractions import Fraction
fracciones = [Fraction(1,2), Fraction(2,3), Fraction(3,4)]
producto = 1
for f in fracciones:
    producto *= f
print(producto)
```

8. Convertir fracción a decimal

```
from fractions import Fraction
f = Fraction(3,4)
print(float(f))
```

9. Convertir decimal a fracción

```
from fractions import Fraction
f = Fraction(0.75)
```

```
print(f)
```

10.Simplificación automática

```
from fractions import Fraction
f = Fraction(6,20)
print(f)
```

11.Comparar fracciones

```
from fractions import Fraction
f1 = Fraction(2,3)
f2 = Fraction(4,6)
print(f1 == f2)
```

12.Sumar y simplificar fracciones con diferente denominador

```
from fractions import Fraction
f1 = Fraction(5,6)
f2 = Fraction(7,8)
print(f1+f2)
```

13.Restar fracciones con diferente denominador

```
from fractions import Fraction
f1 = Fraction(5,6)
f2 = Fraction(7,8)
print(f1-f2)
```

14.Multiplicar fracción por número entero

```
from fractions import Fraction
f = Fraction(3,4)
print(f*5)
```

15.Dividir número entero por fracción

```
from fractions import Fraction
f = Fraction(3,4)
print(5/f)
```

16.Sumar fracción y decimal

```
from fractions import Fraction
f = Fraction(1,2)
d = 0.25
print(f + Fraction(d))
```

17.Restar lista de fracciones

```
from fractions import Fraction
```

```
fracciones = [Fraction(5,6), Fraction(1,2), Fraction(1,3)]
resultado = fracciones[0]
for f in fracciones[1:]:
    resultado -= f
print(resultado)
```

18.Multiplicar varias fracciones

```
from fractions import Fraction
fracciones = [Fraction(1,2), Fraction(3,4), Fraction(2,5)]
producto = 1
for f in fracciones:
    producto *= f
print(producto)
```

19.Simplificar fracción negativa

```
from fractions import Fraction
f = Fraction(-8,12)
print(f)
```

20.Propiedad distributiva

```
from fractions import Fraction
f1 = Fraction(2,3)
f2 = Fraction(1,4)
f3 = Fraction(1,6)
print(f1*(f2+f3) == f1*f2 + f1*f3)
```

---

14. LECCIÓN: PROBLEMAS TIPO SOBRE QUEBRADOS COMUNES

INTRODUCCIÓN

Los **quebrados comunes** son **números fraccionarios** de la forma **numerador/denominador** que representan **partes de un todo**.
Se usan para resolver problemas como:

- Partición de objetos
- Suma y resta de partes
- Distribución equitativa
- Comparaciones de cantidades

Python permite trabajar con quebrados de forma precisa usando:

- **fractions.Fraction**
- Operaciones matemáticas básicas (+, -, *, /)

---

1. CREAR QUEBRADOS COMUNES

```
from fractions import Fraction


q1 = Fraction(3,4)
q2 = Fraction(2,5)
print(q1)
print(q2)
```

Resultado: 3/4 2/5

Explicación:

- `Fraction(a,b)` crea un quebrado **a/b** y lo **simplifica automáticamente**.

---

2. SUMA DE QUEBRADOS

```
suma = q1 + q2
print(suma)
```

Resultado: 23/20

Explicación:

- Se calcula el **MCM de los denominadores** y se suma el numerador correspondiente.

---

3. RESTA DE QUEBRADOS

```
resta = q1 - q2
```

```
print(resta)
```

Resultado: 7/20

---

4. MULTIPLICACIÓN DE QUEBRADOS

```
producto = q1 * q2
print(producto)
```

Resultado: 3/10

Explicación:

- Se multiplican **numeradores entre sí** y **denominadores entre sí**.

---

5. DIVISIÓN DE QUEBRADOS

```
division = q1 / q2
print(division)
```

Resultado: 15/8

Explicación:

- La división se hace multiplicando la primera fracción por el **recíproco** de la segunda.

---

6. CONVERSIÓN ENTRE QUEBRADOS Y DECIMALES

```
print(float(q1))
```

Resultado: 0.75

```
q = Fraction(0.25)
print(q)
```

Resultado: 1/4

---

7. PROBLEMAS COMUNES

1. **Reparto de objetos**

```
from fractions import Fraction
manzanas = Fraction(3,4)
personas = 2
reparto = manzanas / personas
print(reparto)
```

Resultado: 3/8 por persona

2. **Suma de partes de diferentes fracciones**

```
from fractions import Fraction
```

```
q1 = Fraction(1,3)
q2 = Fraction(2,5)
total = q1 + q2
print(total)
```

Resultado: 11/15

3. **Comparación de fracciones**

```
q1 = Fraction(3,4)
q2 = Fraction(5,6)
print(q1 > q2)
```

Resultado: False

---

8. FUNCIONES PARA QUEBRADOS

```
from fractions import Fraction
def sumar(q1,q2):
    return q1+q2

def restar(q1,q2):
    return q1-q2

def multiplicar(q1,q2):
    return q1*q2

def dividir(q1,q2):
    return q1/q2

print(sumar(Fraction(1,2),Fraction(1,3)))
print(restar(Fraction(3,4),Fraction(1,2)))
print(multiplicar(Fraction(2,3),Fraction(3,5)))
print(dividir(Fraction(3,4),Fraction(2,5)))
```

---

9. RESOLVER LISTA DE QUEBRADOS

```
from fractions import Fraction

quebrados = [Fraction(1,2), Fraction(1,3), Fraction(1,4)]
suma_total = sum(quebrados)
producto_total = 1
```

```
for q in quebrados:
    producto_total *= q


print("Suma total:", suma_total)
print("Producto total:", producto_total)
```

Resultado: Suma total 13/12, Producto total 1/24

---

10. RESUMEN GENERAL

- Los **quebrados comunes** representan partes de un todo
- Operaciones: **suma, resta, multiplicación, división**
- Se pueden **simplificar automáticamente** usando fractions.Fraction
- Útiles para resolver problemas de **reparto, suma de partes y comparaciones**

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE PROBLEMAS TIPO SOBRE QUEBRADOS COMUNES

1. Crear un quebrado

```
from fractions import Fraction
q = Fraction(3,4)
print(q)
```

2. Sumar dos quebrados

```
from fractions import Fraction
q1 = Fraction(1,3)
q2 = Fraction(2,5)
print(q1+q2)
```

3. Restar dos quebrados

```
from fractions import Fraction
q1 = Fraction(3,4)
q2 = Fraction(1,2)
print(q1-q2)
```

4. Multiplicar dos quebrados

```
from fractions import Fraction
q1 = Fraction(2,3)
q2 = Fraction(3,5)
print(q1*q2)
```

5. Dividir dos quebrados

```
from fractions import Fraction
q1 = Fraction(3,4)
q2 = Fraction(2,5)
print(q1/q2)
```

6. Comparar quebrados

```
from fractions import Fraction
q1 = Fraction(3,4)
q2 = Fraction(5,6)
print(q1>q2)
```

7. Suma de lista de quebrados

```
from fractions import Fraction
quebrados=[Fraction(1,2),Fraction(1,3),Fraction(1,4)]
print(sum(quebrados))
```

8. Producto de lista de quebrados

```
from fractions import Fraction
quebrados=[Fraction(1,2),Fraction(2,3),Fraction(3,4)]
prod=1
for q in quebrados:
    prod*=q
print(prod)
```

9. Reparto de fracción entre personas

```
from fractions import Fraction
manzanas=Fraction(3,4)
personas=2
print(manzanas/personas)
```

10. Sumar fracción y decimal

```
from fractions import Fraction
q=Fraction(1,3)
d=0.25
print(q+Fraction(d))
```

11. Simplificar fracción automáticamente

```
from fractions import Fraction
q=Fraction(6,20)
print(q)
```

12. Fracción negativa

```
from fractions import Fraction
q=Fraction(-3,4)
print(q)
```

13.Fracción a decimal

```
from fractions import Fraction
q=Fraction(3,4)
print(float(q))
```

14.Comparación de igualdad

```
from fractions import Fraction
q1=Fraction(2,3)
q2=Fraction(4,6)
print(q1==q2)
```

15.Sumar tres quebrados

```
from fractions import Fraction
q1=Fraction(1,2)
q2=Fraction(1,3)
q3=Fraction(1,4)
print(q1+q2+q3)
```

16.Restar tres quebrados

```
from fractions import Fraction
q1=Fraction(5,6)
q2=Fraction(1,2)
q3=Fraction(1,3)
print(q1-q2-q3)
```

17.Multiplicar tres quebrados

```
from fractions import Fraction
q1=Fraction(1,2)
q2=Fraction(2,3)
q3=Fraction(3,4)
print(q1*q2*q3)
```

18.Dividir fracción por fracción negativa

```
from fractions import Fraction
q1=Fraction(3,4)
q2=Fraction(-2,5)
print(q1/q2)
```

19.Sumar y simplificar con diferente denominador

```
from fractions import Fraction
q1=Fraction(7,10)
q2=Fraction(3,5)
print(q1+q2)
```

20.Multiplicar fracción por entero

```
from fractions import Fraction
q=Fraction(3,4)
print(q*5)
```

---

15. LECCIÓN: FRACCIONES CONTINUAS

INTRODUCCIÓN

Las **fracciones continuas** son una forma de representar números como la **suma de un número entero y la fracción recíproca de otra fracción**, de manera anidada:

$$a_0 + 1/(a_1 + 1/(a_2 + 1/(a_3 + ...)))$$

- Se usan para **aproximaciones de números irracionales**
- Permiten expresar **números racionales e irracionales** de forma precisa

En Python, podemos manejar fracciones continuas con:

- **Listas para almacenar los coeficientes a_0, a_1, a_2…**
- Funciones recursivas o iterativas para calcular su valor decimal
- **fractions.Fraction** para mantener exactitud en fracciones

---

1. CREAR FRACCIÓN CONTINUA DE UN NÚMERO RACIONAL

```
from fractions import Fraction

def fraccion_continua(numerador, denominador):
    a = []
    while denominador != 0:
        cociente = numerador // denominador
        a.append(cociente)
        numerador, denominador = denominador, numerador - cociente*denominador
    return a

fc = fraccion_continua(23,7)
print(fc)
```

Resultado: `[3, 1, 2, 3]`

Explicación:

- Se obtiene la **lista de coeficientes** de la fracción continua

---

2. CONVERTIR FRACCIÓN CONTINUA A NÚMERO

```
from fractions import Fraction

def valor_fraccion_continua(a):
```

```
    result = Fraction(a[-1])
    for x in reversed(a[:-1]):
        result = x + 1/result
    return result

fc = [3,1,2,3]
print(valor_fraccion_continua(fc))
```

Resultado: 23/7

Explicación:

- Se reconstruye la fracción sumando el **cociente entero y el recíproco de la siguiente fracción**

---

3. CONVERSIÓN DE NÚMERO DECIMAL A FRACCIÓN CONTINUA

```
from fractions import Fraction

def decimal_a_fc(x, max_iter=10):
    a = []
    while max_iter > 0:
        entero = int(x)
        a.append(entero)
        x -= entero
        if x == 0:
            break
        x = 1/x
        max_iter -= 1
    return a

x = 3.285714
print(decimal_a_fc(x))
```

Resultado aproximado: `[3, 1, 2, 3]`

---

4. PROPIEDADES IMPORTANTES DE FRACCIONES CONTINUAS

- Cada fracción continua es **única para números racionales**
- Permite **aproximar irracionales** con fracciones finitas
- Facilita **operaciones con números grandes o irracionales**

- Las fracciones se pueden simplificar automáticamente con **fractions.Fraction**

---

5. SUMA, RESTA, MULTIPLICACIÓN Y DIVISIÓN (USANDO VALOR DE FRACCIÓN)

```
from fractions import Fraction


fc1 = valor_fraccion_continua([3,1,2,3])
fc2 = valor_fraccion_continua([2,1,3])

suma = fc1 + fc2
resta = fc1 - fc2
producto = fc1 * fc2
division = fc1 / fc2

print(suma, resta, producto, division)
```

Resultado: fracciones simplificadas de cada operación

---

6. SIMPLIFICACIÓN AUTOMÁTICA

```
from fractions import Fraction


f = Fraction(45,14)
print(f)  # 45/14 simplificado
```

---

7. RESUMEN GENERAL

- Las **fracciones continuas** representan números como **una serie de enteros y recíprocos**
- Se pueden construir desde **racionales o decimales**
- Permiten **operaciones exactas usando fractions.Fraction**
- Útiles para **aproximaciones y análisis de números irracionales**

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE FRACCIONES CONTINUAS

1. Crear fracción continua de 23/7

```
from fractions import Fraction
def fraccion_continua(n,d):
    a=[]
```

```
    while d!=0:
        q=n//d
        a.append(q)
        n,d=d,n-q*d
    return a
print(fraccion_continua(23,7))
```

2. Reconstruir fracción desde fracción continua `[3,1,2,3]`

```
from fractions import Fraction
def valor_fc(a):
    r=Fraction(a[-1])
    for x in reversed(a[:-1]):
        r=x+1/r
    return r
print(valor_fc([3,1,2,3]))
```

3. Convertir decimal 3.285714 a fracción continua

```
def decimal_a_fc(x,max_iter=10):
    a=[]
    while max_iter>0:
        e=int(x)
        a.append(e)
        x-=e
        if x==0: break
        x=1/x
        max_iter-=1
    return a
print(decimal_a_fc(3.285714))
```

4. Sumar dos fracciones continuas

```
fc1=valor_fc([3,1,2,3])
fc2=valor_fc([2,1,3])
print(fc1+fc2)
```

5. Restar dos fracciones continuas

```
fc1=valor_fc([3,1,2,3])
fc2=valor_fc([2,1,3])
print(fc1-fc2)
```

6. Multiplicar dos fracciones continuas

```
fc1=valor_fc([3,1,2,3])
```

```
fc2=valor_fc([2,1,3])
print(fc1*fc2)
```

7. Dividir dos fracciones continuas

```
fc1=valor_fc([3,1,2,3])
fc2=valor_fc([2,1,3])
print(fc1/fc2)
```

8. Fracción continua de 7/5

```
print(fraccion_continua(7,5))
```

9. Reconstruir 7/5 desde fracción continua

```
print(valor_fc([1,2,2]))
```

10. Aproximar 0.618033 (phi) a fracción continua

```
print(decimal_a_fc(0.618033,5))
```

11. Sumar fracción continua con entero

```
fc1=valor_fc([3,1,2])
print(fc1+Fraction(1,2))
```

12. Restar fracción continua con entero

```
fc1=valor_fc([3,1,2])
print(fc1-Fraction(2,3))
```

13. Multiplicar fracción continua por entero

```
fc1=valor_fc([3,1,2])
print(fc1*2)
```

14. Dividir fracción continua por entero

```
fc1=valor_fc([3,1,2])
print(fc1/2)
```

15. Reconstruir fracción continua grande

```
fc=[5,1,1,2,3]
print(valor_fc(fc))
```

16. Crear fracción continua de número decimal 2.4

```
print(decimal_a_fc(2.4,5))
```

17. Convertir decimal 3.14159 a fracción continua

```
print(decimal_a_fc(3.14159,10))
```

18. Sumar fracción continua con fracción exacta

```
fc1=valor_fc([1,2,2])
print(fc1+Fraction(3,7))
```

19.Multiplicar fracción continua por fracción exacta

```
fc1=valor_fc([1,2,2])
print(fc1*Fraction(3,5))
```

20.Dividir fracción exacta por fracción continua

```
fc1=valor_fc([1,2,2])
print(Fraction(5,3)/fc1)
```

---

16. LECCIÓN: FRACCIONES DECIMALES

INTRODUCCIÓN

Las **fracciones decimales** son fracciones cuyo **denominador es una potencia de 10**, como **0.5 = 5/10 = 1/2**.
Se utilizan para representar **valores exactos o aproximados** en forma decimal y fraccionaria.

En Python, podemos trabajar con fracciones decimales usando:

- **float** para aproximaciones
- **fractions.Fraction** para exactitud
- Operaciones básicas: **suma, resta, multiplicación, división**

---

1. CREAR FRACCIONES DECIMALES CON FRACTIONS

```
from fractions import Fraction


f1 = Fraction(0.25)
f2 = Fraction(0.75)
print(f1)
print(f2)
```

Resultado:
1/4
3/4

Explicación:

- `Fraction(decimal)` convierte el decimal a **fracción exacta simplificada**

---

2. CONVERTIR DECIMAL A FRACCIÓN

```
from fractions import Fraction


decimal = 0.625
fraccion = Fraction(decimal)
print(fraccion)
```

Resultado: 5/8

Explicación:

- Python representa el decimal como **fracción simplificada con denominador potencia de 2 o 5** cuando es exacta

---

3. OPERACIONES CON FRACCIONES DECIMALES

```
from fractions import Fraction


f1 = Fraction(0.2)
f2 = Fraction(0.3)


suma = f1 + f2
resta = f1 - f2
producto = f1 * f2
division = f1 / f2


print(suma, resta, producto, division)
```

Resultado: 1/2 -1/10 3/50 2/3

Explicación:

- Las fracciones decimales permiten **operaciones exactas sin pérdida de precisión**

---

4. CONVERTIR FRACCIÓN A DECIMAL

```
f = Fraction(3,4)
print(float(f))
```

Resultado: 0.75

Explicación:

- `float()` convierte la fracción decimal a **decimal aproximado**

---

5. SIMPLIFICACIÓN AUTOMÁTICA

```
f = Fraction(50,100)
print(f)
```

Resultado: 1/2

Explicación:

- `Fraction` simplifica automáticamente dividiendo por el **MCD**

---

6. CREAR FUNCIONES PARA FRACCIONES DECIMALES

```
from fractions import Fraction


def sumar(a,b):
    return Fraction(a) + Fraction(b)
```

```
def restar(a,b):
    return Fraction(a) - Fraction(b)

def multiplicar(a,b):
    return Fraction(a) * Fraction(b)

def dividir(a,b):
    return Fraction(a) / Fraction(b)

print(sumar(0.2,0.3))
print(restar(0.7,0.4))
print(multiplicar(0.5,0.6))
print(dividir(0.3,0.2))
```

---

7. USO DE LISTAS DE FRACCIONES DECIMALES

```
from fractions import Fraction

decimales = [0.1, 0.2, 0.25]
fracciones = [Fraction(d) for d in decimales]
suma_total = sum(fracciones)
producto_total = 1
for f in fracciones:
    producto_total *= f

print("Suma total:", suma_total)
print("Producto total:", producto_total)
```

Resultado: Suma total 29/20, Producto total 1/40

---

8. PROPIEDADES IMPORTANTES

- **Exactitud:** permite representar decimales exactos como fracción
- **Simplificación automática:** se obtiene la fracción en su forma más simple
- **Operaciones aritméticas:** suma, resta, multiplicación y división sin errores de redondeo
- Útil para **comparaciones y cálculos precisos**

---

9. RESUMEN GENERAL

- Las **fracciones decimales** representan decimales como **fracción exacta**
- Se crean con **Fraction(decimal)**
- Operaciones: **suma, resta, multiplicación, división**
- Se pueden convertir entre **decimal y fracción**
- Python **simplifica automáticamente**

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE FRACCIONES DECIMALES

1. Convertir 0.25 a fracción

```
from fractions import Fraction
f = Fraction(0.25)
print(f)
```

2. Convertir 0.75 a fracción

```
from fractions import Fraction
f = Fraction(0.75)
print(f)
```

3. Sumar 0.2 y 0.3

```
from fractions import Fraction
f1 = Fraction(0.2)
f2 = Fraction(0.3)
print(f1+f2)
```

4. Restar 0.7 y 0.4

```
from fractions import Fraction
f1 = Fraction(0.7)
f2 = Fraction(0.4)
print(f1-f2)
```

5. Multiplicar 0.5 y 0.6

```
from fractions import Fraction
f1 = Fraction(0.5)
f2 = Fraction(0.6)
print(f1*f2)
```

6. Dividir 0.3 entre 0.2

```
from fractions import Fraction
f1 = Fraction(0.3)
```

```
f2 = Fraction(0.2)
print(f1/f2)
```

7. Convertir 0.625 a fracción

```
from fractions import Fraction
print(Fraction(0.625))
```

8. Convertir fracción 3/4 a decimal

```
from fractions import Fraction
f = Fraction(3,4)
print(float(f))
```

9. Sumar lista de decimales [0.1,0.2,0.25]

```
from fractions import Fraction
decimales=[0.1,0.2,0.25]
fracciones=[Fraction(d) for d in decimales]
print(sum(fracciones))
```

10. Multiplicar lista de decimales [0.1,0.2,0.25]

```
from fractions import Fraction
decimales=[0.1,0.2,0.25]
fracciones=[Fraction(d) for d in decimales]
prod=1
for f in fracciones: prod*=f
print(prod)
```

11. Simplificar fracción 50/100

```
from fractions import Fraction
f=Fraction(50,100)
print(f)
```

12. Comparar 0.5 y 1/2

```
from fractions import Fraction
f1=Fraction(0.5)
f2=Fraction(1,2)
print(f1==f2)
```

13. Sumar decimal y fracción

```
from fractions import Fraction
print(Fraction(0.25)+Fraction(1,4))
```

14. Restar decimal y fracción

```
from fractions import Fraction
```

```
print(Fraction(0.5)-Fraction(1,3))
```

15.Multiplicar decimal y fracción

```
from fractions import Fraction
print(Fraction(0.2)*Fraction(2,5))
```

16.Dividir decimal y fracción

```
from fractions import Fraction
print(Fraction(0.3)/Fraction(3,10))
```

17.Convertir lista de decimales a fracciones

```
from fractions import Fraction
decimales=[0.2,0.4,0.6]
fracciones=[Fraction(d) for d in decimales]
print(fracciones)
```

18.Sumar fracciones decimales en bucle

```
from fractions import Fraction
decimales=[0.1,0.15,0.2]
total=Fraction(0)
for d in decimales: total+=Fraction(d)
print(total)
```

19.Multiplicar fracciones decimales en bucle

```
from fractions import Fraction
decimales=[0.1,0.2,0.5]
prod=1
for d in decimales: prod*=Fraction(d)
print(prod)
```

20.Operaciones mixtas

```
from fractions import Fraction
f1=Fraction(0.3)
f2=Fraction(2,5)
print("Suma:",f1+f2)
print("Resta:",f1-f2)
print("Multiplicación:",f1*f2)
print("División:",f1/f2)
```

---

17. LECCIÓN: CONVERSIÓN DE FRACCIONES

INTRODUCCIÓN

La **conversión de fracciones** consiste en transformar una fracción de una forma a otra:

- **Decimal** → Fracción exacta
- **Fracción** → Decimal
- **Fracción impropia** → Número mixto
- **Número mixto** → Fracción impropia

Python permite hacer estas conversiones de manera precisa usando:

- **fractions.Fraction** para fracciones exactas
- Operaciones matemáticas básicas (+, -, *, /)
- Funciones personalizadas para convertir entre formas

---

1. CONVERTIR DECIMAL A FRACCIÓN

```
from fractions import Fraction


decimal = 0.75
fraccion = Fraction(decimal)
print(fraccion)
```

Resultado: 3/4

Explicación:

- `Fraction(decimal)` convierte el decimal a **fracción exacta simplificada**

---

2. CONVERTIR FRACCIÓN A DECIMAL

```
from fractions import Fraction


f = Fraction(3,4)
decimal = float(f)
print(decimal)
```

Resultado: 0.75

Explicación:

- `float()` convierte la fracción a decimal aproximado

---

3. CONVERTIR FRACCIÓN IMPROPIA A NÚMERO MIXTO

```
from fractions import Fraction


f = Fraction(7,4)
entero = f.numerator // f.denominator
resto = f.numerator % f.denominator
print(f"{entero} {resto}/{f.denominator}")
```

Resultado: 1 3/4

Explicación:

- Se divide el numerador entre el denominador para obtener el **entero** y el **resto**

---

4. CONVERTIR NÚMERO MIXTO A FRACCIÓN IMPROPIA

```
entero = 1
numerador = 3
denominador = 4


f = Fraction(entero*denominador + numerador, denominador)
print(f)
```

Resultado: 7/4

---

5. SUMA Y RESTA CON CONVERSIÓN AUTOMÁTICA

```
from fractions import Fraction


f1 = Fraction(1,2)
f2 = Fraction(3,4)

suma = f1 + f2
resta = f1 - f2


print(suma, resta)
```

Resultado: 5/4 -1/4

---

6. MULTIPLICACIÓN Y DIVISIÓN CON FRACCIONES

```
producto = f1 * f2
division = f1 / f2
```

```
print(producto, division)
```

Resultado: 3/8 2/3

Explicación:

- Multiplicación: **numerador x numerador, denominador x denominador**
- División: **multiplicar por recíproco**

---

7. USO DE FUNCIONES PARA CONVERSIÓN

```
from fractions import Fraction

def decimal_a_fraccion(d):
    return Fraction(d)

def fraccion_a_decimal(f):
    return float(f)

def impropia_a_mixta(f):
    entero = f.numerator // f.denominator
    resto = f.numerator % f.denominator
    return f"{entero} {resto}/{f.denominator}"

def mixta_a_impropia(entero, numerador, denominador):
    return Fraction(entero*denominador + numerador, denominador)

print(decimal_a_fraccion(0.6))
print(fraccion_a_decimal(Fraction(3,5)))
print(impropia_a_mixta(Fraction(9,4)))
print(mixta_a_impropia(2,1,3))
```

---

8. RESUMEN GENERAL

- **Conversiones comunes:**
    - Decimal ↔ Fracción
    - Fracción impropia ↔ Número mixto
- Usar **fractions.Fraction** para exactitud
- Las operaciones entre fracciones se simplifican automáticamente

- Convertir a decimal usando **float()**

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE CONVERSIÓN DE FRACCIONES

1. Decimal 0.25 a fracción

```
from fractions import Fraction
print(Fraction(0.25))
```

2. Decimal 0.75 a fracción

```
from fractions import Fraction
print(Fraction(0.75))
```

3. Fracción 3/4 a decimal

```
from fractions import Fraction
print(float(Fraction(3,4)))
```

4. Fracción impropia 7/4 a número mixto

```
f=Fraction(7,4)
e=f.numerator//f.denominator
r=f.numerator%f.denominator
print(f"{e} {r}/{f.denominator}")
```

5. Número mixto 1 3/4 a fracción impropia

```
from fractions import Fraction
e,n,d=1,3,4
print(Fraction(e*d+n,d))
```

6. Sumar 1/2 + 3/4 con conversión

```
from fractions import Fraction
f1=Fraction(1,2)
f2=Fraction(3,4)
print(f1+f2)
```

7. Restar 3/4 - 1/2

```
from fractions import Fraction
f1=Fraction(3,4)
f2=Fraction(1,2)
print(f1-f2)
```

8. Multiplicar 1/2 * 3/4

```
from fractions import Fraction
f1=Fraction(1,2)
```

```
f2=Fraction(3,4)
print(f1*f2)
```

9. Dividir 1/2 / 3/4

```
from fractions import Fraction
f1=Fraction(1,2)
f2=Fraction(3,4)
print(f1/f2)
```

10. Decimal 0.6 a fracción

```
from fractions import Fraction
print(Fraction(0.6))
```

11. Fracción 3/5 a decimal

```
from fractions import Fraction
print(float(Fraction(3,5)))
```

12. Fracción impropia 9/4 a número mixto

```
f=Fraction(9,4)
e=f.numerator//f.denominator
r=f.numerator%f.denominator
print(f"{e} {r}/{f.denominator}")
```

13. Número mixto 2 1/3 a fracción impropia

```
from fractions import Fraction
print(Fraction(2*3+1,3))
```

14. Sumar fracción y decimal 0.5 + 1/3

```
from fractions import Fraction
print(Fraction(0.5)+Fraction(1,3))
```

15. Restar decimal y fracción 0.7 - 2/5

```
from fractions import Fraction
print(Fraction(0.7)-Fraction(2,5))
```

16. Multiplicar decimal y fracción 0.4 * 3/5

```
from fractions import Fraction
print(Fraction(0.4)*Fraction(3,5))
```

17. Dividir decimal y fracción 0.6 / 2/3

```
from fractions import Fraction
print(Fraction(0.6)/Fraction(2,3))
```

18. Convertir lista de decimales a fracciones

```
from fractions import Fraction
```

```
dec=[0.1,0.2,0.25]
fracs=[Fraction(d) for d in dec]
print(fracs)
```

19.Sumar lista de fracciones

```
from fractions import Fraction
dec=[0.1,0.2,0.25]
fracs=[Fraction(d) for d in dec]
print(sum(fracs))
```

20.Multiplicar lista de fracciones

```
from fractions import Fraction
dec=[0.1,0.2,0.25]
fracs=[Fraction(d) for d in dec]
prod=1
for f in fracs: prod*=f
print(prod)
```

---

18. LECCIÓN: SISTEMA MÉTRICO DECIMAL

INTRODUCCIÓN

El **sistema métrico decimal** es un sistema de **medición basado en múltiplos de 10**, utilizado para **longitud, masa, capacidad, área y volumen**.
Python permite trabajar con conversiones y cálculos de este sistema de manera sencilla:

- Operaciones aritméticas: **+**, **-**, *****, **/**
- Funciones personalizadas para **convertir unidades**
- Uso de **diccionarios o listas** para almacenar equivalencias

---

1. CONCEPTO DE UNIDADES MÉTRICAS

- **Longitud:** milímetro (mm), centímetro (cm), metro (m), kilómetro (km)
- **Masa:** miligramo (mg), gramo (g), kilogramo (kg), tonelada (t)
- **Capacidad/Volumen:** mililitro (ml), litro (l), decilitro (dl)

Relaciones importantes:

- 1 m = 100 cm = 1000 mm
- 1 kg = 1000 g = 1,000,000 mg
- 1 l = 1000 ml

---

2. CONVERTIR ENTRE UNIDADES

```
# Convertir metros a centímetros
metros = 2.5
centimetros = metros * 100
print(centimetros)
```

Resultado: 250

Explicación:

- Se multiplica por la **potencia de 10** correspondiente

```
# Convertir gramos a kilogramos
gramos = 1500
kilogramos = gramos / 1000
print(kilogramos)
```

Resultado: 1.5

---

3. FUNCIONES PARA CONVERSIONES

```
def metros_a_centimetros(m):
    return m * 100

def centimetros_a_metros(cm):
    return cm / 100

def gramos_a_kilogramos(g):
    return g / 1000

def kilogramos_a_gramos(kg):
    return kg * 1000

print(metros_a_centimetros(3.2))
print(centimetros_a_metros(250))
print(gramos_a_kilogramos(1500))
print(kilogramos_a_gramos(2.5))
```

---

4. CONVERSIONES CON DICCIONARIOS

```
# Diccionario de múltiplos métricos
longitud = {'mm':0.001,'cm':0.01,'m':1,'km':1000}

def convertir_longitud(valor, de, a):
    return valor * longitud[de] / longitud[a]

print(convertir_longitud(1500,'m','km'))  # 1.5 km
print(convertir_longitud(2,'km','m'))     # 2000 m
```

---

5. OPERACIONES CON UNIDADES MÉTRICAS

```
# Suma de longitudes en diferentes unidades
l1 = 2.5   # metros
l2 = 150   # centímetros

# Convertir l2 a metros antes de sumar
l2_m = l2 / 100
total = l1 + l2_m
```

```
print(total)
```

Resultado: 4.0 m

---

6. RESUMEN DE OPERACIONES

- Para **convertir unidades** multiplicar o dividir por **potencias de 10**
- Para **operaciones entre distintas unidades**, convertir primero a **una unidad común**
- Se pueden usar **funciones o diccionarios** para automatizar conversiones

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE SISTEMA MÉTRICO DECIMAL

1. Convertir 3 metros a centímetros

```
m=3
print(m*100)
```

2. Convertir 2500 cm a metros

```
cm=2500
print(cm/100)
```

3. Convertir 1500 gramos a kilogramos

```
g=1500
print(g/1000)
```

4. Convertir 2.5 kg a gramos

```
kg=2.5
print(kg*1000)
```

5. Convertir 5000 ml a litros

```
ml=5000
print(ml/1000)
```

6. Convertir 3.2 litros a mililitros

```
l=3.2
print(l*1000)
```

7. Convertir 2 km a metros

```
km=2
print(km*1000)
```

8. Convertir 4500 m a km

```
m=4500
print(m/1000)
```

9. Sumar 2.5 m + 150 cm

```
l1=2.5
l2=150/100
print(l1+l2)
```

10. Restar 3 km - 500 m

```
km=3
m=500/1000
print(km-m)
```

11. Multiplicar 2 m * 3

```
m=2
print(m*3)
```

12. Dividir 5 km entre 2

```
km=5
print(km/2)
```

13. Convertir 750 mg a gramos

```
mg=750
print(mg/1000)
```

14. Convertir 0.5 t a kg

```
t=0.5
print(t*1000)
```

15. Convertir 1200 ml a litros

```
ml=1200
print(ml/1000)
```

16. Convertir 7 km a cm

```
km=7
print(km*1000*100)
```

17. Convertir 350 cm a mm

```
cm=350
print(cm*10)
```

18. Sumar 2.5 kg + 750 g

```
kg=2.5
g=750/1000
print(kg+g)
```

19. Restar 5 l - 2500 ml

```
l=5
```

```
ml=2500/1000
print(l-ml)
```

20.Convertir 1.2 t a g

```
t=1.2
print(t*1000*1000)
```

---

19. LECCIÓN: DENSIDAD

INTRODUCCIÓN

La **densidad** es una propiedad física que relaciona la **masa de un cuerpo con su volumen**.
Se define como:

Densidad = Masa / Volumen

- **Unidades:**
    - Masa: gramos (g), kilogramos (kg)
    - Volumen: centímetros cúbicos (cm^3), metros cúbicos (m^3), litros (l)
    - Densidad: g/cm^3, kg/m^3, kg/l

En Python, podemos:

- Calcular densidad
- Calcular masa o volumen a partir de densidad
- Convertir unidades fácilmente
- Automatizar cálculos mediante funciones

---

1. CALCULAR DENSIDAD

```
masa = 500  # gramos
volumen = 250  # cm³

densidad = masa / volumen
print(densidad)
```

Resultado: 2.0 g/cm^3

Explicación:

- La densidad se obtiene dividiendo **masa entre volumen**

---

2. CALCULAR MASA A PARTIR DE DENSIDAD

```
densidad = 2.5  # g/cm³
volumen = 200  # cm³

masa = densidad * volumen
print(masa)
```

Resultado: 500 g

---

3. CALCULAR VOLUMEN A PARTIR DE DENSIDAD

```
masa = 800  # g
densidad = 2  # g/cm³

volumen = masa / densidad
print(volumen)
```

Resultado: 400 cm³

---

4. FUNCIONES PARA DENSIDAD

```
def calcular_densidad(masa, volumen):
    return masa / volumen

def calcular_masa(densidad, volumen):
    return densidad * volumen

def calcular_volumen(masa, densidad):
    return masa / densidad

print(calcular_densidad(500,250))
print(calcular_masa(2.5,200))
print(calcular_volumen(800,2))
```

---

5. CONVERSIÓN DE UNIDADES

```
# Convertir masa de gramos a kilogramos
masa_g = 500
masa_kg = masa_g / 1000
print(masa_kg)

# Convertir volumen de cm³ a m³
volumen_cm3 = 250000
volumen_m3 = volumen_cm3 / 1e6
print(volumen_m3)
```

---

6. OPERACIONES CON DENSIDAD

```
# Promedio de densidades de varios objetos
densidades = [2.0, 1.5, 3.0]
```

```
promedio = sum(densidades) / len(densidades)
print(promedio)
```

---

7. RESUMEN

- **Densidad = Masa / Volumen**
- Calcular masa: **Masa = Densidad × Volumen**
- Calcular volumen: **Volumen = Masa / Densidad**
- Usar **funciones** para automatizar cálculos
- Convertir unidades según necesidad

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE DENSIDAD

1. Calcular densidad de 500 g y 250 cm³

```
masa=500
volumen=250
print(masa/volumen)
```

2. Calcular masa de densidad 2.5 g/cm³ y volumen 200 cm³

```
densidad=2.5
volumen=200
print(densidad*volumen)
```

3. Calcular volumen de masa 800 g y densidad 2 g/cm³

```
masa=800
densidad=2
print(masa/densidad)
```

4. Función densidad de 600 g y 300 cm³

```
def densidad(m,v): return m/v
print(densidad(600,300))
```

5. Función masa con densidad 3 g/cm³ y volumen 150 cm³

```
def masa(d,v): return d*v
print(masa(3,150))
```

6. Función volumen con masa 900 g y densidad 1.5 g/cm³

```
def volumen(m,d): return m/d
print(volumen(900,1.5))
```

7. Convertir masa 750 g a kg

```
masa_g=750
print(masa_g/1000)
```

8. Convertir volumen 500000 cm³ a m³

```
vol=500000
print(vol/1e6)
```

9. Calcular densidad promedio de 2.0, 3.0, 1.5 g/cm³

```
densidades=[2.0,3.0,1.5]
print(sum(densidades)/len(densidades))
```

10. Calcular masa con densidad 0.8 g/cm³ y volumen 500 cm³

```
d=0.8
v=500
print(d*v)
```

11. Calcular volumen con masa 1200 g y densidad 1.2 g/cm³

```
m=1200
d=1.2
print(m/d)
```

12. Calcular densidad de 450 g y 150 cm³

```
m=450
v=150
print(m/v)
```

13. Convertir 2.5 kg a g

```
kg=2.5
print(kg*1000)
```

14. Convertir 0.003 m³ a cm³

```
m3=0.003
print(m3*1e6)
```

15. Sumar densidades 1.2 + 2.5 + 3.3

```
dens=[1.2,2.5,3.3]
print(sum(dens))
```

16. Promedio de densidades 1.5, 2.0, 2.5

```
dens=[1.5,2.0,2.5]
print(sum(dens)/len(dens))
```

17. Calcular masa de densidad 1.8 y volumen 400

```
d=1.8
v=400
print(d*v)
```

18. Calcular volumen de masa 720 y densidad 2.4

```
m=720
d=2.4
print(m/d)
```

19.Convertir masa 0.75 kg a g

```
kg=0.75
print(kg*1000)
```

20.Convertir volumen 2.5 m^3 a cm^3

```
m3=2.5
print(m3*1e6)
```

---

20. LECCIÓN: OTROS SISTEMAS DE MEDICIÓN

INTRODUCCIÓN

Además del **sistema métrico decimal**, existen otros sistemas de medición como:

- **Sistema Inglés o Imperial:** pulgadas (in), pies (ft), yardas (yd), millas (mi), libras (lb), onzas (oz), galones (gal)
- **Sistema de Temperatura:** Celsius (°C), Fahrenheit (°F), Kelvin (K)
- **Sistema de Tiempo:** horas, minutos, segundos
- **Sistema de Área:** acres, pies², yardas²

Python permite:

- **Convertir unidades** entre distintos sistemas
- Realizar **operaciones aritméticas** entre distintas unidades
- Crear **funciones o diccionarios** para automatizar conversiones

---

1. CONVERSIONES DE LONGITUD (IMPERIAL ↔ MÉTRICO)

```
# 1 pulgada = 2.54 cm
pulgadas = 12
cm = pulgadas * 2.54
print(cm)

# 1 pie = 12 pulgadas
pies = 5
pulgadas_totales = pies * 12
print(pulgadas_totales)

# 1 milla = 1.60934 km
millas = 3
km = millas * 1.60934
print(km)
```

---

2. CONVERSIONES DE MASA (IMPERIAL ↔ MÉTRICO)

```
# 1 libra = 0.453592 kg
libras = 10
kg = libras * 0.453592
print(kg)
```

```
# 1 onza = 28.3495 g
onzas = 16
g = onzas * 28.3495
print(g)
```

---

3. CONVERSIONES DE VOLUMEN

```
# 1 galón = 3.78541 litros
gal = 5
litros = gal * 3.78541
print(litros)

# 1 pinta = 0.473176 litros
pinta = 8
litros_pinta = pinta * 0.473176
print(litros_pinta)
```

---

4. CONVERSIONES DE TEMPERATURA

```
# Celsius a Fahrenheit
def celsius_a_fahrenheit(c):
    return (c * 9/5) + 32

# Fahrenheit a Celsius
def fahrenheit_a_celsius(f):
    return (f - 32) * 5/9

# Celsius a Kelvin
def celsius_a_kelvin(c):
    return c + 273.15

print(celsius_a_fahrenheit(25))
print(fahrenheit_a_celsius(77))
print(celsius_a_kelvin(25))
```

---

5. CONVERSIONES DE ÁREA

```
# 1 acre = 4046.86 m²
```

```
acres = 3
m2 = acres * 4046.86
print(m2)

# 1 pie² = 0.092903 m²
pies2 = 500
m2_pies = pies2 * 0.092903
print(m2_pies)
```

---

6. FUNCIONES PARA CONVERSIONES AUTOMÁTICAS

```
def pulgadas_a_cm(p):
    return p * 2.54

def pies_a_m(p):
    return p * 0.3048

def millas_a_km(mi):
    return mi * 1.60934

def libras_a_kg(lb):
    return lb * 0.453592

def onzas_a_g(oz):
    return oz * 28.3495

def gal_a_l(gal):
    return gal * 3.78541

def pinta_a_l(p):
    return p * 0.473176

print(pulgadas_a_cm(12))
print(pies_a_m(5))
print(millas_a_km(3))
print(libras_a_kg(10))
print(onzas_a_g(16))
```

```
print(gal_a_l(5))
print(pinta_a_l(8))
```

---

7. RESUMEN GENERAL

- **Otros sistemas de medición** incluyen: imperial, temperatura, área
- **Conversiones** se realizan multiplicando por el **factor de conversión adecuado**
- Se recomienda usar **funciones o diccionarios** para automatizar las conversiones
- Python permite trabajar con **operaciones exactas** y resultados en unidades deseadas

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE OTROS SISTEMAS DE MEDICIÓN

1. Convertir 12 pulgadas a cm

```
print(12*2.54)
```

2. Convertir 5 pies a pulgadas

```
print(5*12)
```

3. Convertir 3 millas a km

```
print(3*1.60934)
```

4. Convertir 10 libras a kg

```
print(10*0.453592)
```

5. Convertir 16 onzas a gramos

```
print(16*28.3495)
```

6. Convertir 5 galones a litros

```
print(5*3.78541)
```

7. Convertir 8 pintas a litros

```
print(8*0.473176)
```

8. Celsius 25 °C a Fahrenheit

```
print((25*9/5)+32)
```

9. Fahrenheit 77 °F a Celsius

```
print((77-32)*5/9)
```

10. Celsius 25 °C a Kelvin

```
print(25+273.15)
```

11. Convertir 3 acres a m²

```
print(3*4046.86)
```

12. Convertir 500 pies² a m²

```
print(500*0.092903)
```

13.Convertir 20 pulgadas a cm usando función

```
def pulg_cm(p): return p*2.54
print(pulg_cm(20))
```

14.Convertir 10 pies a m usando función

```
def pies_m(p): return p*0.3048
print(pies_m(10))
```

15.Convertir 5 millas a km usando función

```
def mi_km(mi): return mi*1.60934
print(mi_km(5))
```

16.Convertir 25 libras a kg usando función

```
def lb_kg(lb): return lb*0.453592
print(lb_kg(25))
```

17.Convertir 32 onzas a g usando función

```
def oz_g(oz): return oz*28.3495
print(oz_g(32))
```

18.Convertir 2 galones a litros usando función

```
def gal_l(gal): return gal*3.78541
print(gal_l(2))
```

19.Convertir 6 pintas a litros usando función

```
def pinta_l(p): return p*0.473176
print(pinta_l(6))
```

20.Convertir 10 pies² a m² usando función

```
def pies2_m2(p): return p*0.092903
print(pies2_m2(10))
```

---

21. LECCIÓN: ÁREAS DE FIGURAS PLANAS Y VOLÚMENES DE CUERPOS GEOMÉTRICOS

INTRODUCCIÓN

En **geometría**, calcular **áreas de figuras planas** y **volúmenes de cuerpos geométricos** es fundamental.
Python permite:

- Calcular **áreas**: cuadrado, rectángulo, triángulo, círculo, trapecio
- Calcular **volúmenes**: cubo, prisma, cilindro, esfera, cono
- Usar **funciones** para automatizar cálculos
- Aplicar **módulo math** para operaciones con π y raíces

---

1. CÁLCULO DE ÁREAS DE FIGURAS PLANAS

```
import math


# Cuadrado
lado = 5
area_cuadrado = lado ** 2
print(area_cuadrado)

# Rectángulo
base = 4
altura = 6
area_rectangulo = base * altura
print(area_rectangulo)

# Triángulo
base = 6
altura = 8
area_triangulo = (base * altura) / 2
print(area_triangulo)

# Círculo
radio = 3
area_circulo = math.pi * radio ** 2
print(area_circulo)
```

```
# Trapecio
base_mayor = 8
base_menor = 5
altura = 4
area_trapecio = ((base_mayor + base_menor) * altura) / 2
print(area_trapecio)
```

---

2. CÁLCULO DE VOLÚMENES DE CUERPOS GEOMÉTRICOS

```
# Cubo
lado = 4
vol_cubo = lado ** 3
print(vol_cubo)

# Prisma rectangular
largo = 5
ancho = 3
alto = 2
vol_prisma = largo * ancho * alto
print(vol_prisma)

# Cilindro
radio = 3
altura = 7
vol_cilindro = math.pi * radio ** 2 * altura
print(vol_cilindro)

# Esfera
radio = 5
vol_esfera = (4/3) * math.pi * radio ** 3
print(vol_esfera)

# Cono
radio = 3
altura = 6
vol_cono = (1/3) * math.pi * radio ** 2 * altura
```

```
print(vol_cono)
```

---

3. FUNCIONES PARA ÁREAS Y VOLÚMENES

```
import math


def area_cuadrado(lado): return lado**2

def area_rectangulo(base, altura): return base*altura

def area_triangulo(base, altura): return (base*altura)/2

def area_circulo(radio): return math.pi*radio**2

def area_trapecio(base_mayor, base_menor, altura): return
((base_mayor+base_menor)*altura)/2


def vol_cubo(lado): return lado**3

def vol_prisma(largo, ancho, alto): return largo*ancho*alto

def vol_cilindro(radio, altura): return math.pi*radio**2*altura

def vol_esfera(radio): return (4/3)*math.pi*radio**3

def vol_cono(radio, altura): return (1/3)*math.pi*radio**2*altura


print(area_cuadrado(5))

print(area_rectangulo(4,6))

print(area_triangulo(6,8))

print(area_circulo(3))

print(area_trapecio(8,5,4))

print(vol_cubo(4))

print(vol_prisma(5,3,2))

print(vol_cilindro(3,7))

print(vol_esfera(5))

print(vol_cono(3,6))
```

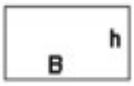
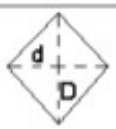
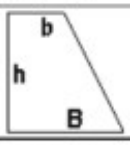
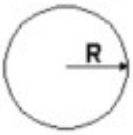
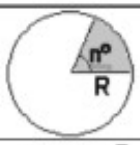
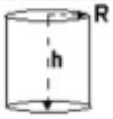
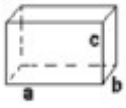
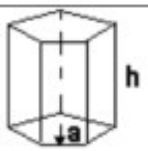
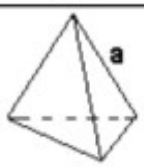
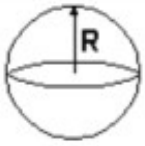
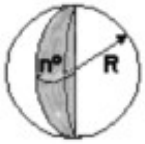
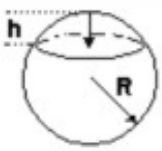
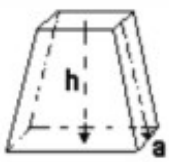
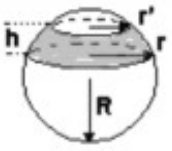
---

4. RESUMEN

- **Áreas de figuras planas:** cuadrado, rectángulo, triángulo, círculo, trapecio
- **Volúmenes de cuerpos geométricos:** cubo, prisma, cilindro, esfera, cono
- Usar **fórmulas geométricas estándar**
- Usar **math.pi** para cálculos con π
- Crear **funciones** facilita la reutilización y automatización

---

TABLA DE AREAS Y VOLUMENES

	Cuadrado $A = a^2$	Triángulo $A = \frac{B \cdot h}{2}$	
	Rectángulo $A = B \cdot h$	Romboide $A = B \cdot h$	
	Rombo $A = \frac{D \cdot d}{2}$	Trapecio $A = \frac{(B+b) \cdot h}{2}$	
	Polígono regular $A = \frac{P \cdot a}{2}$ **a** es la apotema / **P** es el perímetro (suma de la longitud de los lados)	Círculo $P = 2\pi \cdot R$ $A = \pi \cdot R^2$	
	Corona circular $A = \pi \cdot (R^2 - r^2)$	Sector circular $A = \frac{\pi \cdot R^2 \cdot n}{360}$	
	Cubo $A = 6a^2$ $V = a^3$	Cilindro $A = 2\pi \cdot R(h+R)$ $V = \pi \cdot R^2 \cdot h$	
	Ortoedro $A = 2 \cdot (ab + ac + bc)$ $V = a \cdot b \cdot c$	Prisma recto $A = P \cdot (h + a)$ $V = A_B \cdot h$ A_B es área de la base	
g es la generatriz	Cono $A = \pi \cdot R^2 (h + g)$ $V = \frac{\pi \cdot R^2 \cdot h}{3}$	Tronco de cono $A = p \cdot \left[\left(g \cdot (r + R) + r_2 + R^2 \right) \right]$ $V = \frac{p \cdot h \cdot (R^2 + r^2 + Rr)}{3}$	
	Tetraedro regular $A = \sqrt{3} \cdot a^2$ $V = \frac{\sqrt{2} \cdot a^2}{12}$	Esfera $A = 4 \cdot \pi \cdot R^2$ $V = \frac{4 \cdot \pi \cdot R^3}{3}$	
	Pirámide recta $A = \frac{P \cdot (a + a')}{2}$ $V = \frac{A_B \cdot h}{3}$	Huso: cuña esférica $A = \frac{4\pi \cdot R^2 \cdot n}{360}$ $V = \frac{V_E \cdot n}{360}$	
	Octaedro regular $A = 2\sqrt{3} \cdot a^2$ $V = \frac{\sqrt{2} \cdot a^3}{3}$	Casquete esférico $A = 2\pi \cdot R \cdot h$ $V = \frac{\pi \cdot h^2 \cdot (3R - h)}{3}$	
	Tronco de pirámide $A = \frac{(P + P') \cdot a \cdot A_B + A'_B}{2}$ $V = \frac{\left(A_B + A'_B + \sqrt{A_B \cdot A'_B} \right) \cdot h}{3}$	Zona esférica $A = 2\pi \cdot R \cdot h$ $V = \frac{\pi \cdot h \cdot \left(h^2 + 3r^2 + 3r'^2 \right)}{6}$	

20 PROGRAMAS NUMERADOS Y RESUELTOS: ÁREAS Y VOLÚMENES

1. Área de cuadrado lado 6

```
print(6**2)
```

2. Área de rectángulo base 5, altura 8

```
print(5*8)
```

3. Área de triángulo base 10, altura 6

```
print((10*6)/2)
```

4. Área de círculo radio 4

```
import math
print(math.pi*4**2)
```

5. Área de trapecio bases 10 y 6, altura 5

```
print((10+6)*5/2)
```

6. Volumen de cubo lado 3

```
print(3**3)
```

7. Volumen de prisma largo 4, ancho 3, alto 5

```
print(4*3*5)
```

8. Volumen de cilindro radio 3, altura 7

```
import math
print(math.pi*3**2*7)
```

9. Volumen de esfera radio 6

```
import math
print((4/3)*math.pi*6**3)
```

10. Volumen de cono radio 3, altura 9

```
import math
print((1/3)*math.pi*3**2*9)
```

11. Función área cuadrado lado 8

```
def ac(l): return l**2
print(ac(8))
```

12. Función área rectángulo 7x5

```
def ar(b,h): return b*h
print(ar(7,5))
```

13. Función área triángulo 9x6

```
def at(b,h): return (b*h)/2
print(at(9,6))
```

14.Función área círculo radio 5

```
import math
def acirc(r): return math.pi*r**2
print(acirc(5))
```

15.Función área trapecio bases 8 y 4 altura 6

```
def atrap(bm,bn,h): return (bm+bn)*h/2
print(atrap(8,4,6))
```

16.Función volumen cubo lado 4

```
def vc(l): return l**3
print(vc(4))
```

17.Función volumen prisma 6x3x2

```
def vp(l,a,h): return l*a*h
print(vp(6,3,2))
```

18.Función volumen cilindro radio 2 altura 5

```
import math
def vcyl(r,h): return math.pi*r**2*h
print(vcyl(2,5))
```

19.Función volumen esfera radio 3

```
import math
def vesf(r): return (4/3)*math.pi*r**3
print(vesf(3))
```

20.Función volumen cono radio 3 altura 4

```
import math
def vcono(r,h): return (1/3)*math.pi*r**2*h
print(vcono(3,4))
```

---

22. LECCIÓN: TEOREMA DE PITÁGORAS

INTRODUCCIÓN

El **Teorema de Pitágoras** es una de las bases fundamentales de la **geometría y trigonometría**. Establece que en un **triángulo rectángulo**, el **cuadrado de la hipotenusa** es igual a la **suma de los cuadrados de los catetos**.

Fórmula general:
$c^2 = a^2 + b^2$

Donde:

- **a** y **b** son los catetos.
- **c** es la hipotenusa.

Python permite aplicar este teorema fácilmente mediante **operaciones aritméticas**, el uso de la **librería math** y **funciones personalizadas**.

---

1. CÁLCULO DE LA HIPOTENUSA

Para hallar la **hipotenusa**, se aplica la raíz cuadrada de la suma de los cuadrados de los catetos.

```
import math

a = 3

b = 4

c = math.sqrt(a**2 + b**2)

print("Hipotenusa:", c)
```

Resultado: 5.0

---

2. CÁLCULO DE UN CATETO CONOCIDA LA HIPOTENUSA

Si conocemos la hipotenusa y un cateto, podemos calcular el otro cateto con:
$a = \sqrt{(c^2 - b^2)}$

```
import math

c = 13

b = 5

a = math.sqrt(c**2 - b**2)

print("Cateto faltante:", a)
```

Resultado: 12.0

---

3. COMPROBACIÓN DEL TEOREMA

Podemos comprobar si tres lados forman un triángulo rectángulo verificando que:
$c^2 = a^2 + b^2$

```
a, b, c = 5, 12, 13
if c**2 == a**2 + b**2:
    print("Es un triángulo rectángulo")
else:
    print("No cumple el teorema de Pitágoras")
```

---

4. USO DE FUNCIONES PARA GENERALIZAR EL CÁLCULO

```
import math
def hipotenusa(a, b):
    return math.sqrt(a**2 + b**2)

print("Hipotenusa de (6,8):", hipotenusa(6,8))
```

---

5. APLICACIÓN GEOMÉTRICA

El teorema se aplica para:

- Calcular **distancias** entre dos puntos (distancia euclidiana).
- Determinar la **longitud de diagonales** en figuras geométricas.
- Problemas de **física y navegación** donde intervienen trayectorias rectas.

---

6. CÁLCULO DE DISTANCIA ENTRE DOS PUNTOS (USANDO PITÁGORAS)

```
import math
x1, y1 = (2, 3)
x2, y2 = (7, 8)
distancia = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
print("Distancia entre los puntos:", distancia)
```

---

7. REDONDEO DE RESULTADOS

Podemos redondear la hipotenusa a un número determinado de decimales usando **round()**:

```
import math
a, b = 3, 4
c = math.sqrt(a**2 + b**2)
print("Hipotenusa redondeada:", round(c, 2))
```

---

8. VISUALIZACIÓN CON MÓDULO MATPLOTLIB (OPCIONAL)

```
import matplotlib.pyplot as plt
```

```
a = 3
b = 4
plt.plot([0, a], [0, 0], 'b', label='Cateto a')
plt.plot([a, a], [0, b], 'r', label='Cateto b')
plt.plot([0, a], [0, b], 'g', label='Hipotenusa c')
plt.legend()
plt.grid(True)
plt.show()
```

RESUMEN DE PASOS

1. Representar los lados del triángulo.
2. Aplicar **$c = \sqrt{(a^2 + b^2)}$** para la hipotenusa.
3. Aplicar **$a = \sqrt{(c^2 - b^2)}$** o **$b = \sqrt{(c^2 - a^2)}$** para el cateto.
4. Verificar la **igualdad pitagórica**.
5. Aplicar el teorema en distancias y diagonales.

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE TEOREMA DE PITÁGORAS

1. Calcular hipotenusa de un triángulo (3,4)

```
import math; print(math.sqrt(3**2+4**2))
```

2. Calcular cateto faltante (hipotenusa=13, cateto=5)

```
import math; print(math.sqrt(13**2-5**2))
```

3. Verificar triángulo rectángulo

```
a,b,c=8,15,17; print(c**2==a**2+b**2)
```

4. Hipotenusa con valores flotantes

```
import math; print(round(math.sqrt(2.5**2+3.5**2),2))
```

5. Calcular distancia entre dos puntos

```
import math; print(math.sqrt((7-2)**2+(9-3)**2))
```

6. Función para hipotenusa

```
import math; h=lambda a,b:math.sqrt(a**2+b**2); print(h(5,12))
```

7. Calcular cateto menor

```
import math; print(math.sqrt(10**2-6**2))
```

8. Comprobación del teorema

```
a,b,c=9,40,41; print("Cumple" if c**2==a**2+b**2 else "No cumple")
```

9. Redondear resultado

```
import math; print(round(math.sqrt(9**2+12**2),3))
```

10. Diagonal de un cuadrado

```
import math; lado=5; print(math.sqrt(lado**2+lado**2))
```

11. Diagonal de un rectángulo

```
import math; a,b=6,8; print(math.sqrt(a**2+b**2))
```

12. Cateto en triángulo isósceles rectángulo

```
import math; c=10; print(math.sqrt((c**2)/2))
```

13. Distancia en coordenadas negativas

```
import math; print(math.sqrt((-4-3)**2+(-5-1)**2))
```

14. Verificar triángulo no rectángulo

```
a,b,c=4,5,6; print(c**2==a**2+b**2)
```

15. Calcular hipotenusa con entrada del usuario

```
import math; a=float(input()); b=float(input()); print(math.sqrt(a**2+b**2))
```

16. Calcular cateto según valores del usuario

```
import math; c=float(input()); b=float(input()); print(math.sqrt(c**2-b**2))
```

17. Generar tabla de hipotenusas

```
import math
for a in range(1,6):
    for b in range(1,6):
        print(a,b,round(math.sqrt(a**2+b**2),2))
```

18. Verificar varias ternas pitagóricas

```
ternas=[(3,4,5),(5,12,13),(7,24,25)]
for a,b,c in ternas: print(c**2==a**2+b**2)
```

19. Calcular distancia diagonal 3D

```
import math; print(round(math.sqrt(3**2+4**2+12**2),2))
```

20. Calcular hipotenusa con valores aleatorios

```
import math,random; a,b=random.randint(1,10),random.randint(1,10);
print(a,b,math.sqrt(a**2+b**2))
```

Esta lección te enseña **cómo aplicar el Teorema de Pitágoras en Python** para cálculos geométricos, medición de distancias, y generación de triángulos rectángulos.
¿Quieres que te genere una **versión PDF imprimible** con todos los ejemplos resueltos y explicaciones paso a paso?

23. LECCIÓN: LONGITUD Y TIEMPO

INTRODUCCIÓN

En programación y física, trabajar con **longitud** y **tiempo** es fundamental.
Python permite:

- **Convertir unidades** de longitud y tiempo
- Realizar **operaciones aritméticas** con distintas unidades
- Automatizar conversiones usando **funciones**
- Aplicar **factores de conversión** para pasar de un sistema a otro

---

1. CONCEPTOS DE LONGITUD

- **Sistema Métrico:** milímetro (mm), centímetro (cm), metro (m), kilómetro (km)
- **Sistema Imperial:** pulgadas (in), pies (ft), yardas (yd), millas (mi)
- **Relaciones importantes:**
    - 1 m = 100 cm = 1000 mm
    - 1 km = 1000 m
    - 1 pulgada = 2.54 cm
    - 1 pie = 12 pulgadas
    - 1 yarda = 3 pies
    - 1 milla = 1.60934 km

---

2. CONVERSIÓN DE LONGITUD

```
# Métrico
metros = 5
cm = metros * 100
km = metros / 1000
print(cm, km)

# Imperial
pulgadas = 12
cm2 = pulgadas * 2.54
pies = pulgadas / 12
print(cm2, pies)
```

---

3. FUNCIONES PARA LONGITUD

```
def m_a_cm(m): return m*100
def cm_a_m(cm): return cm/100
def m_a_km(m): return m/1000
def pulg_a_cm(p): return p*2.54
def pies_a_m(ft): return ft*0.3048
def millas_a_km(mi): return mi*1.60934

print(m_a_cm(7))
print(cm_a_m(350))
print(m_a_km(1500))
print(pulg_a_cm(10))
print(pies_a_m(5))
print(millas_a_km(3))
```

---

4. CONCEPTOS DE TIEMPO

- **Unidades comunes:** segundos (s), minutos (min), horas (h), días
- **Relaciones:**
    - 1 min = 60 s
    - 1 h = 60 min = 3600 s
    - 1 día = 24 h = 86400 s

---

5. CONVERSIONES DE TIEMPO

```
# Segundos a minutos y horas
segundos = 7200
minutos = segundos / 60
horas = segundos / 3600
print(minutos, horas)

# Minutos a segundos
minutos2 = 45
segundos2 = minutos2 * 60
print(segundos2)

# Horas a segundos
```

```
horas2 = 3
segundos3 = horas2 * 3600
print(segundos3)
```

---

6. FUNCIONES PARA TIEMPO

```
def seg_a_min(s): return s/60
def seg_a_h(s): return s/3600
def min_a_seg(m): return m*60
def h_a_seg(h): return h*3600
def h_a_min(h): return h*60

print(seg_a_min(180))
print(seg_a_h(7200))
print(min_a_seg(45))
print(h_a_seg(3))
print(h_a_min(2))
```

---

7. RESUMEN

- **Longitud:** m, cm, km, in, ft, yd, mi
- **Tiempo:** s, min, h, días
- Convertir usando **factores de conversión**
- Crear **funciones** para facilitar cálculos y automatizar conversiones
- Python permite combinar conversiones y operaciones aritméticas fácilmente

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE LONGITUD Y TIEMPO

1. Convertir 7 m a cm

```
print(7*100)
```

2. Convertir 350 cm a m

```
print(350/100)
```

3. Convertir 1500 m a km

```
print(1500/1000)
```

4. Convertir 10 pulgadas a cm

```
print(10*2.54)
```

5. Convertir 5 pies a m

```
print(5*0.3048)
```

6. Convertir 3 millas a km

```
print(3*1.60934)
```

7. Segundos 1800 a minutos

```
print(1800/60)
```

8. Segundos 7200 a horas

```
print(7200/3600)
```

9. Minutos 45 a segundos

```
print(45*60)
```

10. Horas 3 a segundos

```
print(3*3600)
```

11. Horas 2 a minutos

```
print(2*60)
```

12. Función metros a cm, 12 m

```
def m_cm(m): return m*100
print(m_cm(12))
```

13. Función cm a metros, 500 cm

```
def cm_m(cm): return cm/100
print(cm_m(500))
```

14. Función millas a km, 5 mi

```
def mi_km(mi): return mi*1.60934
print(mi_km(5))
```

15. Función pies a m, 8 ft

```
def ft_m(ft): return ft*0.3048
print(ft_m(8))
```

16. Función segundos a minutos, 360 s

```
def s_min(s): return s/60
print(s_min(360))
```

17. Función segundos a horas, 10800 s

```
def s_h(s): return s/3600
print(s_h(10800))
```

18. Función minutos a segundos, 90 min

```
def min_s(m): return m*60
print(min_s(90))
```

19. Función horas a segundos, 5 h

```
def h_s(h): return h*3600
print(h_s(5))
```

20.Función horas a minutos, 7 h

```
def h_min(h): return h*60
print(h_min(7))
```

---

24. LECCIÓN: RAZONES Y PROPORCIONES

INTRODUCCIÓN

Las **razones y proporciones** son conceptos matemáticos fundamentales que permiten **comparar cantidades** y resolver problemas de **proporcionalidad**.
En Python se pueden:

- Calcular **razones** como fracciones o decimales
- Resolver **proporciones** mediante **regla de tres**
- Crear **funciones** para automatizar cálculos
- Trabajar con números **enteros o decimales**

---

1. CONCEPTO DE RAZÓN

- Una **razón** es la comparación de dos cantidades mediante **división**:

Razón = a / b

```
a = 10
b = 5
razon = a / b
print(razon)
```

Resultado: 2.0

- Se puede expresar también como **a:b**

---

2. PROPORCIONES

- Una **proporción** es una igualdad entre **dos razones**:

a / b = c / d

- Se puede resolver mediante **producto cruzado**:

a * d = b * c

```
a = 4
b = 5
c = 8
# Resolver d
d = (b * c) / a
print(d)
```

Resultado: 10.0

---

3. FUNCIONES PARA RAZONES Y PROPORCIONES

```
# Calcular razón
def calcular_razon(a,b):
    return a/b

# Resolver proporción a/b = c/d
def resolver_proporcion(a,b,c):
    return (b*c)/a

print(calcular_razon(10,5))
print(resolver_proporcion(4,5,8))
```

---

4. USO DE DECIMALES Y FRACCIONES

```
from fractions import Fraction

# Razón como fracción
a = 3
b = 4
razon = Fraction(a,b)
print(razon)

# Proporción con fracción
a=2
b=3
c=8
d = Fraction(b*c,a)
print(d)
```

---

5. APLICACIONES PRÁCTICAS

- Ajustar recetas
- Escalas en mapas o planos
- Comparar cantidades de distintos objetos

```
# Ejemplo: receta proporcional
harina = 200  # gramos
azucar = 100  # gramos
# Si se quiere duplicar la receta
```

```
harina2 = harina*2
azucar2 = azucar*2
print(harina2, azucar2)
```

---

6. RESUMEN

- **Razón = a / b**
- **Proporción = igualdad de dos razones a/b = c/d**
- Resolver proporciones usando **producto cruzado**
- Python permite usar **decimales o fracciones**
- Se pueden crear **funciones** para automatizar cálculos y resolver problemas

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE RAZONES Y PROPORCIONES

1. Calcular razón 10:5

```
print(10/5)
```

2. Calcular razón 3:4

```
print(3/4)
```

3. Calcular razón 7:2

```
print(7/2)
```

4. Resolver proporción 4/5 = 8/d

```
print(5*8/4)
```

5. Resolver proporción 2/3 = 10/d

```
print(3*10/2)
```

6. Resolver proporción 5/7 = 20/d

```
print(7*20/5)
```

7. Función calcular razón 12 y 4

```
def r(a,b): return a/b
print(r(12,4))
```

8. Función resolver proporción 6/8 = 12/d

```
def p(a,b,c): return b*c/a
print(p(6,8,12))
```

9. Razón como fracción 3/5

```
from fractions import Fraction
print(Fraction(3,5))
```

10.Proporción con fracciones 2/3 = 8/d

```
from fractions import Fraction
print(Fraction(3*8,2))
```

11.Duplicar receta harina 150, azúcar 100

```
harina,azucar=150,100
print(harina*2, azucar*2)
```

12.Triplicar receta harina 200, azúcar 50

```
harina,azucar=200,50
print(harina*3, azucar*3)
```

13.Reducir a la mitad receta harina 300, azúcar 150

```
harina,azucar=300,150
print(harina/2, azucar/2)
```

14.Calcular razón 9:12

```
print(9/12)
```

15.Resolver proporción 7/9 = 14/d

```
print(9*14/7)
```

16.Calcular razón 15:5

```
print(15/5)
```

17.Resolver proporción 8/12 = 20/d

```
print(12*20/8)
```

18.Razón con fracción 7/10

```
from fractions import Fraction
print(Fraction(7,10))
```

19.Proporción con fracción 5/8 = 15/d

```
from fractions import Fraction
print(Fraction(8*15,5))
```

20.Ajuste proporcional: receta 250g harina, 100g azúcar al triple

```
harina,azucar=250,100
print(harina*3, azucar*3)
```

---

25. LECCIÓN: INTERÉS SIMPLE

INTRODUCCIÓN

El **interés simple** es un concepto financiero que se utiliza para calcular la **ganancia o costo adicional** sobre un capital inicial durante un **período de tiempo determinado**.
Python permite:

- Calcular **interés simple**
- Calcular **capital inicial** o **tiempo** a partir de interés
- Crear **funciones** para automatizar cálculos
- Usar operaciones aritméticas básicas

---

1. FÓRMULA DEL INTERÉS SIMPLE

Fórmula principal:

I = C × r × t

Donde:

- **I** = interés
- **C** = capital inicial
- **r** = tasa de interés (en decimal, ej: 5% → 0.05)
- **t** = tiempo (en años, meses, según convención)

```
C = 1000  # Capital en dólares
r = 0.05  # Tasa de interés 5%
t = 2     # Tiempo en años

I = C * r * t
print(I)
```

Resultado: 100.0

---

2. CALCULAR CAPITAL A PARTIR DEL INTERÉS

```
I = 200
r = 0.04
t = 3

C = I / (r * t)
print(C)
```

Resultado: 1666.6667

---

3. CALCULAR TIEMPO A PARTIR DEL INTERÉS

```
I = 150
C = 500
r = 0.05

t = I / (C * r)
print(t)
```

Resultado: 0.6 años

---

4. FUNCIONES PARA INTERÉS SIMPLE

```
def interes_simple(C, r, t):
    return C * r * t

def capital(I, r, t):
    return I / (r * t)

def tiempo(I, C, r):
    return I / (C * r)

print(interes_simple(1000, 0.05, 2))
print(capital(200, 0.04, 3))
print(tiempo(150, 500, 0.05))
```

---

5. CALCULAR MONTO TOTAL

- El **monto total** (capital + interés) se calcula como:

M = C + I

```
C = 1000
r = 0.05
t = 2

I = C * r * t
M = C + I
print(M)
```

Resultado: 1100

---

6. RESUMEN

- **Interés simple = C × r × t**
- **Capital = I / (r × t)**
- **Tiempo = I / (C × r)**
- **Monto total = C + I**
- Usar **funciones** para automatizar cálculos

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE INTERÉS SIMPLE

1. Calcular interés simple: 1000, 5%, 2 años

```
print(1000*0.05*2)
```

2. Calcular interés simple: 500, 3%, 4 años

```
print(500*0.03*4)
```

3. Calcular capital a partir de interés: I=200, r=4%, t=3

```
print(200/(0.04*3))
```

4. Calcular tiempo a partir de interés: I=150, C=500, r=5%

```
print(150/(500*0.05))
```

5. Calcular monto total: C=1000, r=5%, t=2

```
C=1000; r=0.05; t=2
I=C*r*t
print(C+I)
```

6. Función interés simple, 2000, 6%, 3 años

```
def I(C,r,t): return C*r*t
print(I(2000,0.06,3))
```

7. Función capital a partir de interés: I=300, r=4%, t=2

```
def C(I,r,t): return I/(r*t)
print(C(300,0.04,2))
```

8. Función tiempo: I=120, C=400, r=5%

```
def t(I,C,r): return I/(C*r)
print(t(120,400,0.05))
```

9. Calcular interés: C=800, r=0.03, t=5

```
print(800*0.03*5)
```

10. Calcular capital: I=90, r=0.06, t=2

```
print(90/(0.06*2))
```

11.Calcular tiempo: I=50, C=500, r=0.02

```
print(50/(500*0.02))
```

12.Calcular monto total: C=1500, r=0.04, t=3

```
C=1500; r=0.04; t=3
I=C*r*t
print(C+I)
```

13.Interés simple: C=600, r=0.05, t=1.5

```
print(600*0.05*1.5)
```

14.Capital a partir de interés: I=75, r=0.03, t=1

```
print(75/(0.03*1))
```

15.Tiempo a partir de interés: I=180, C=900, r=0.04

```
print(180/(900*0.04))
```

16.Monto total: C=2500, r=0.06, t=2

```
C=2500; r=0.06; t=2
I=C*r*t
print(C+I)
```

17.Interés simple: C=1200, r=0.07, t=1

```
print(1200*0.07*1)
```

18.Capital: I=210, r=0.05, t=2

```
print(210/(0.05*2))
```

19.Tiempo: I=360, C=800, r=0.09

```
print(360/(800*0.09))
```

20.Monto total: C=500, r=0.08, t=4

```
C=500; r=0.08; t=4
I=C*r*t
print(C+I)
```

---

26. LECCIÓN: INTERÉS COMPUESTO

INTRODUCCIÓN

El **interés compuesto** es un concepto financiero que calcula el **interés sobre el capital inicial más los intereses acumulados** en períodos anteriores.
Python permite:

- Calcular **interés compuesto** y **monto total**
- Determinar **capital inicial**, **tasa** o **tiempo** si se conocen los demás valores
- Crear **funciones** para automatizar los cálculos
- Usar operaciones matemáticas con **potencias**

---

1. FÓRMULA DEL INTERÉS COMPUESTO

Fórmula principal:

M = C × (1 + r)^t

Donde:

- **M** = monto total (capital + interés)
- **C** = capital inicial
- **r** = tasa de interés por período (decimal, ej: 5% → 0.05)
- **t** = número de períodos

El **interés compuesto** se obtiene restando el capital inicial:

I = M - C

```
C = 1000  # Capital inicial
r = 0.05  # Tasa de interés 5%
t = 3     # Períodos


M = C * (1 + r)**t
I = M - C
print(M, I)
```

Resultado: 1157.625 157.625

---

2. CALCULAR CAPITAL INICIAL A PARTIR DEL MONTO

```
M = 1157.625
r = 0.05
t = 3
```

```
C = M / (1 + r)**t
print(C)
```

Resultado: 1000.0

---

3. CALCULAR TIEMPO A PARTIR DEL MONTO

```
import math

M = 1576.25
C = 1000
r = 0.05

t = math.log(M / C) / math.log(1 + r)
print(t)
```

Resultado: 9.0 períodos

---

4. FUNCIONES PARA INTERÉS COMPUESTO

```
import math

def monto_compuesto(C, r, t):
    return C * (1 + r)**t

def interes_compuesto(C, r, t):
    return C * (1 + r)**t - C

def capital_inicial(M, r, t):
    return M / (1 + r)**t

def tiempo_compuesto(M, C, r):
    return math.log(M / C) / math.log(1 + r)

print(monto_compuesto(1000,0.05,3))
print(interes_compuesto(1000,0.05,3))
print(capital_inicial(1157.625,0.05,3))
print(tiempo_compuesto(1576.25,1000,0.05))
```

---

5. MONTO TOTAL CON INTERÉS COMPUESTO MENSUAL

- Si la tasa es anual pero los períodos son **mensuales**:

M = C × (1 + r/n)^(n*t)

Donde **n** = número de períodos por año

```
C = 1000
r = 0.12  # 12% anual
n = 12    # mensual
t = 2     # años


M = C * (1 + r/n)**(n*t)
print(M)
```

Resultado: 1270.244

---

6. RESUMEN

- **Interés compuesto = C × (1 + r)^t**
- **Interés = M - C**
- Para calcular **capital o tiempo** usar fórmulas inversas y logaritmos
- Python permite usar **math.log** y **potencias**
- Crear **funciones** facilita automatizar cálculos financieros

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE INTERÉS COMPUESTO

1. Monto compuesto C=1000, r=5%, t=3

```
print(1000*(1+0.05)**3)
```

2. Interés compuesto C=1000, r=5%, t=3

```
print(1000*(1+0.05)**3-1000)
```

3. Capital inicial M=1157.625, r=5%, t=3

```
print(1157.625/(1+0.05)**3)
```

4. Tiempo compuesto M=1576.25, C=1000, r=5%

```
import math
print(math.log(1576.25/1000)/math.log(1+0.05))
```

5. Monto compuesto anual C=2000, r=6%, t=5

```
print(2000*(1+0.06)**5)
```

6. Interés compuesto anual C=2000, r=6%, t=5

```
print(2000*(1+0.06)**5-2000)
```

7. Capital inicial M=2676.22, r=6%, t=5

```
print(2676.22/(1+0.06)**5)
```

8. Tiempo compuesto M=3000, C=2000, r=8%

```
import math
print(math.log(3000/2000)/math.log(1+0.08))
```

9. Monto compuesto mensual C=1500, r=12%, n=12, t=2

```
print(1500*(1+0.12/12)**(12*2))
```

10. Interés compuesto mensual C=1500, r=12%, n=12, t=2

```
print(1500*(1+0.12/12)**(12*2)-1500)
```

11. Capital inicial mensual M=1800, r=10%, n=12, t=1

```
print(1800/(1+0.10/12)**(12*1))
```

12. Tiempo compuesto mensual M=2200, C=2000, r=12%, n=12

```
import math
print(math.log(2200/2000)/(12*math.log(1+0.12/12)))
```

13. Monto compuesto C=500, r=7%, t=4

```
print(500*(1+0.07)**4)
```

14. Interés compuesto C=500, r=7%, t=4

```
print(500*(1+0.07)**4-500)
```

15. Capital inicial M=650, r=7%, t=4

```
print(650/(1+0.07)**4)
```

16. Tiempo compuesto M=1000, C=600, r=5%

```
import math
print(math.log(1000/600)/math.log(1+0.05))
```

17. Monto compuesto C=1200, r=8%, t=3

```
print(1200*(1+0.08)**3)
```

18. Interés compuesto C=1200, r=8%, t=3

```
print(1200*(1+0.08)**3-1200)
```

19. Capital inicial M=1520, r=8%, t=3

```
print(1520/(1+0.08)**3)
```

20. Tiempo compuesto M=2000, C=1500, r=6%

```
import math
print(math.log(2000/1500)/math.log(1+0.06))
```

---

27. LECCIÓN: AMORTIZACIÓN

INTRODUCCIÓN

La **amortización** es un concepto financiero que se refiere a **la distribución de pagos de un préstamo o deuda a lo largo del tiempo**, incluyendo capital e intereses.
Python permite:

- Calcular **cuotas de amortización**
- Determinar **saldo pendiente** y **interés por período**
- Usar **funciones** para automatizar cálculos
- Aplicar **fórmulas de amortización fija (sistema francés)** y **decreciente**

---

1. AMORTIZACIÓN SISTEMA FRANCÉS (CUOTA FIJA)

Fórmula de la cuota:

A = C × (r(1+r)^n) / ((1+r)^n - 1)

Donde:

- **A** = cuota periódica
- **C** = capital inicial
- **r** = tasa de interés por período (decimal)
- **n** = número de períodos

```
C = 10000  # Capital
r = 0.05   # Tasa por período
n = 12     # Número de períodos (meses)

A = C * (r*(1+r)**n) / ((1+r)**n - 1)
print(A)
```

---

2. CALCULAR AMORTIZACIÓN MENSUAL CON CICLO

```
C = 10000
r = 0.05
n = 12

A = C * (r*(1+r)**n) / ((1+r)**n - 1)
saldo = C

for i in range(1, n+1):
```

```
    interes = saldo * r
    capital_pagado = A - interes
    saldo -= capital_pagado
    print(f"Mes {i}: Cuota={A:.2f}, Interés={interes:.2f},
Capital={capital_pagado:.2f}, Saldo={saldo:.2f}")
```

---

3. FUNCIONES PARA AMORTIZACIÓN

```
def cuota_fija(C,r,n):
    return C*(r*(1+r)**n)/((1+r)**n - 1)


def amortizacion_frances(C,r,n):
    A = cuota_fija(C,r,n)
    saldo = C
    tabla = []
    for i in range(1,n+1):
        interes = saldo*r
        capital_pagado = A - interes
        saldo -= capital_pagado
        tabla.append((i,A,interes,capital_pagado,saldo))
    return tabla


tabla = amortizacion_frances(10000,0.05,12)
for fila in tabla:
    print(f"Mes {fila[0]}: Cuota={fila[1]:.2f}, Interés={fila[2]:.2f},
Capital={fila[3]:.2f}, Saldo={fila[4]:.2f}")
```

---

4. AMORTIZACIÓN SISTEMA DECRECIENTE

- En este sistema **el capital amortizado es constante**, y la cuota disminuye con el tiempo:

```
C = 10000
n = 12
capital_amortizado = C/n
r = 0.05
saldo = C

for i in range(1,n+1):
    interes = saldo*r
    cuota = capital_amortizado + interes
```

```
    saldo -= capital_amortizado
    print(f"Mes {i}: Cuota={cuota:.2f}, Interés={interes:.2f},
Capital={capital_amortizado:.2f}, Saldo={saldo:.2f}")
```

---

5. RESUMEN

- **Amortización:** distribución de pagos de un préstamo
- **Sistema francés:** cuota fija, interés disminuye, capital aumenta
- **Sistema decreciente:** capital constante, cuota disminuye
- Usar **funciones** y **bucles** para generar tablas de amortización
- Python facilita calcular **intereses**, **cuotas** y **saldo** automáticamente

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE AMORTIZACIÓN

1. Calcular cuota fija: C=10000, r=5%, n=12

```
C=10000; r=0.05; n=12
A = C*(r*(1+r)**n)/((1+r)**n -1)
print(A)
```

2. Tabla amortización francesa C=5000, r=4%, n=6

```
C=5000; r=0.04; n=6
A = C*(r*(1+r)**n)/((1+r)**n -1)
saldo = C
for i in range(1,n+1):
    interes=saldo*r
    capital=A-interes
    saldo-=capital
    print(f"Mes {i}: Cuota={A:.2f}, Interés={interes:.2f},
Capital={capital:.2f}, Saldo={saldo:.2f}")
```

3. Amortización decreciente C=6000, r=3%, n=6

```
C=6000; r=0.03; n=6
capital=C/n
saldo=C
for i in range(1,n+1):
    interes=saldo*r
    cuota=capital+interes
    saldo-=capital
    print(f"Mes {i}: Cuota={cuota:.2f}, Interés={interes:.2f},
Capital={capital:.2f}, Saldo={saldo:.2f}")
```

4. Función cuota fija C=8000, r=6%, n=10

```
def cuota(C,r,n): return C*(r*(1+r)**n)/((1+r)**n -1)
print(cuota(8000,0.06,10))
```

5. Función tabla francesa C=7000, r=5%, n=12

```
def tabla_francesa(C,r,n):
    A=C*(r*(1+r)**n)/((1+r)**n-1)
    saldo=C
    for i in range(1,n+1):
        interes=saldo*r
        capital=A-interes
        saldo-=capital
        print(f"Mes {i}: Cuota={A:.2f}, Interés={interes:.2f},
Capital={capital:.2f}, Saldo={saldo:.2f}")
tabla_francesa(7000,0.05,12)
```

6. Calcular saldo después de 5 meses C=10000, r=5%, n=12

```
C=10000; r=0.05; n=12
A=C*(r*(1+r)**n)/((1+r)**n-1)
saldo=C
for i in range(1,6):
    interes=saldo*r
    capital=A-interes
    saldo-=capital
print(saldo)
```

7. Amortización decreciente con saldo final C=9000, r=4%, n=9

```
C=9000; r=0.04; n=9
capital=C/n
saldo=C
for i in range(1,n+1):
    interes=saldo*r
    cuota=capital+interes
    saldo-=capital
    print(f"Mes {i}: Cuota={cuota:.2f}, Interés={interes:.2f},
Capital={capital:.2f}, Saldo={saldo:.2f}")
```

8. Función amortización decreciente

```
def tabla_decreciente(C,r,n):
    capital=C/n
```

```
    saldo=C
    for i in range(1,n+1):
        interes=saldo*r
        cuota=capital+interes
        saldo-=capital
        print(f"Mes {i}: Cuota={cuota:.2f}, Interés={interes:.2f},
Capital={capital:.2f}, Saldo={saldo:.2f}")
tabla_decreciente(12000,0.05,12)
```

9. Cuota fija mensual C=15000, r=3%, n=12

```
C=15000; r=0.03; n=12
A=C*(r*(1+r)**n)/((1+r)**n-1)
print(A)
```

10. Interés primer mes francés C=15000, r=3%, n=12

```
C=15000; r=0.03; n=12
A=C*(r*(1+r)**n)/((1+r)**n-1)
interes=C*r
print(interes)
```

11. Capital pagado primer mes C=15000, r=3%, n=12

```
C=15000; r=0.03; n=12
A=C*(r*(1+r)**n)/((1+r)**n-1)
interes=C*r
capital=A-interes
print(capital)
```

12. Saldo después de 3 meses francés

```
C=10000; r=0.05; n=12
A=C*(r*(1+r)**n)/((1+r)**n-1)
saldo=C
for i in range(1,4):
    interes=saldo*r
    capital=A-interes
    saldo-=capital
print(saldo)
```

13. Cuota mensual decreciente C=10000, r=4%, n=12

```
C=10000; r=0.04; n=12
capital=C/n
saldo=C
```

```
interes=saldo*r
cuota=capital+interes
print(cuota)
```

14.Tabla completa francesa C=20000, r=5%, n=12

```
C=20000; r=0.05; n=12
A=C*(r*(1+r)**n)/((1+r)**n-1)
saldo=C
for i in range(1,n+1):
    interes=saldo*r
    capital=A-interes
    saldo-=capital
    print(f"Mes {i}: Cuota={A:.2f}, Interés={interes:.2f},
Capital={capital:.2f}, Saldo={saldo:.2f}")
```

15.Tabla completa decreciente C=20000, r=5%, n=12

```
C=20000; r=0.05; n=12
capital=C/n
saldo=C
for i in range(1,n+1):
    interes=saldo*r
    cuota=capital+interes
    saldo-=capital
    print(f"Mes {i}: Cuota={cuota:.2f}, Interés={interes:.2f},
Capital={capital:.2f}, Saldo={saldo:.2f}")
```

16.Calcular primera cuota francés C=8000, r=4%, n=10

```
C=8000; r=0.04; n=10
A=C*(r*(1+r)**n)/((1+r)**n-1)
print(A)
```

17.Calcular primera cuota decreciente C=8000, r=4%, n=10

```
C=8000; r=0.04; n=10
capital=C/n
interes=C*r
cuota=capital+interes
print(cuota)
```

18.Saldo final después de 6 meses francés

```
C=10000; r=0.05; n=12
A=C*(r*(1+r)**n)/((1+r)**n-1)
```

```
saldo=C
for i in range(1,7):
    interes=saldo*r
    capital=A-interes
    saldo-=capital
print(saldo)
```

19.Saldo final después de 6 meses decreciente

```
C=10000; r=0.05; n=12
capital=C/n
saldo=C
for i in range(1,7):
    interes=saldo*r
    cuota=capital+interes
    saldo-=capital
print(saldo)
```

20.Función completa francesa con impresión de tabla

```
def amort_francesa(C,r,n):
    A=C*(r*(1+r)**n)/((1+r)**n-1)
    saldo=C
    for i in range(1,n+1):
        interes=saldo*r
        capital=A-interes
        saldo-=capital
        print(f"Mes {i}: Cuota={A:.2f}, Interés={interes:.2f},
Capital={capital:.2f}, Saldo={saldo:.2f}")
amort_francesa(15000,0.05,12)
```

---

28. LECCIÓN: DESCUENTO

INTRODUCCIÓN

El **descuento** es un concepto financiero que indica la **reducción del precio original de un producto o servicio**, generalmente expresado en porcentaje.
Python permite:

- Calcular **precio con descuento**
- Calcular **monto de descuento**
- Crear **funciones** para automatizar cálculos
- Aplicar cálculos a **listas de productos** o cantidades

---

1. FÓRMULA DEL DESCUENTO

Monto de descuento:

D = P × d

Precio final:

PF = P - D

Donde:

- **P** = precio original
- **d** = tasa de descuento (decimal, ej: 20% → 0.20)
- **D** = monto del descuento
- **PF** = precio final después del descuento

```
P = 500  # Precio original
d = 0.20 # Descuento 20%


D = P * d
PF = P - D
print(D, PF)
```

Resultado: 100 400

---

2. CALCULAR DESCUENTO EN PORCENTAJE

Si conocemos **precio original y precio final**, se puede calcular la tasa de descuento:

d = (P - PF) / P

```
P = 500
PF = 400
```

```
d = (P - PF) / P
print(d)
```

Resultado: 0.20

---

3. FUNCIONES PARA DESCUENTO

```
def monto_descuento(P,d):
    return P*d

def precio_final(P,d):
    return P - P*d

def tasa_descuento(P,PF):
    return (P-PF)/P

print(monto_descuento(500,0.20))
print(precio_final(500,0.20))
print(tasa_descuento(500,400))
```

---

4. DESCUENTO MÚLTIPLE

- Para aplicar **varios descuentos sucesivos**:

```
P = 1000
descuentos = [0.10, 0.05, 0.02]  # 10%, 5%, 2%

PF = P
for d in descuentos:
    PF = PF * (1 - d)

print(PF)
```

Resultado: 833.10

---

5. DESCUENTO EN LISTA DE PRODUCTOS

```
productos = [500, 1200, 800]
descuento = 0.15

precios_finales = [p*(1-descuento) for p in productos]
```

```
print(precios_finales)
```

Resultado: [425.0, 1020.0, 680.0]

---

6. RESUMEN

- **Monto de descuento = P × d**
- **Precio final = P - D**
- **Tasa de descuento = (P - PF) / P**
- Python permite usar **bucles, listas y funciones** para automatizar cálculos de descuento
- Se pueden aplicar **descuentos sucesivos** de forma acumulativa

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE DESCUENTO

1. Calcular descuento 500 con 20%

```
print(500*0.20)
```

2. Precio final 500 con 20%

```
print(500-500*0.20)
```

3. Calcular tasa de descuento P=500, PF=400

```
print((500-400)/500)
```

4. Descuento 1200 con 15%

```
print(1200*0.15)
```

5. Precio final 1200 con 15%

```
print(1200-1200*0.15)
```

6. Tasa de descuento P=800, PF=680

```
print((800-680)/800)
```

7. Función monto descuento P=1000, d=10%

```
def D(P,d): return P*d
print(D(1000,0.10))
```

8. Función precio final P=1000, d=10%

```
def PF(P,d): return P-P*d
print(PF(1000,0.10))
```

9. Función tasa de descuento P=1000, PF=850

```
def d(P,PF): return (P-PF)/P
print(d(1000,850))
```

10. Descuento sucesivo 1000 con 10%,5%,2%

```
P=1000
```

```
des=[0.10,0.05,0.02]
for d in des: P=P*(1-d)
print(P)
```

11.Lista de productos [500, 1000, 1500] con 20%

```
prods=[500,1000,1500]
desc=0.20
print([p*(1-desc) for p in prods])
```

12.Calcular descuento 750 con 12%

```
print(750*0.12)
```

13.Precio final 750 con 12%

```
print(750-750*0.12)
```

14.Tasa de descuento P=600, PF=480

```
print((600-480)/600)
```

15.Función monto descuento P=900, d=8%

```
def D(P,d): return P*d
print(D(900,0.08))
```

16.Función precio final P=900, d=8%

```
def PF(P,d): return P-P*d
print(PF(900,0.08))
```

17.Lista productos [200,400,600] con 5%

```
prods=[200,400,600]
desc=0.05
print([p*(1-desc) for p in prods])
```

18.Descuento sucesivo 500 con 10%,5%

```
P=500
for d in [0.10,0.05]: P=P*(1-d)
print(P)
```

19.Calcular tasa descuento P=1200, PF=900

```
print((1200-900)/1200)
```

20.Precio final de 1500 con 25%

```
print(1500-1500*0.25)
```

---

29. LECCIÓN: REPARTO PROPORCIONAL

INTRODUCCIÓN

El **reparto proporcional** es una técnica para distribuir una cantidad total entre varias partes de forma **proporcional a ciertos valores o criterios**.
Python permite:

- Calcular **participación proporcional**
- Automatizar el cálculo con **listas** y **bucles**
- Crear **funciones** para distribuir montos
- Aplicar repartos a **dinero, recursos o cantidades físicas**

---

1. FÓRMULA DEL REPARTO PROPORCIONAL

Si queremos repartir **M** entre varias partes con valores **v1, v2, …, vn**, la **parte proporcional** de cada valor se calcula como:

Pi = M × (vi / S)

Donde:

- **Pi** = parte proporcional del valor **vi**
- **M** = monto total a repartir
- **vi** = valor individual de cada parte
- **S = v1 + v2 + … + vn** = suma total de los valores

```
M = 1000
valores = [2, 3, 5]


S = sum(valores)
partes = [M*(v/S) for v in valores]
print(partes)
```

Resultado: [200.0, 300.0, 500.0]

---

2. FUNCIONES PARA REPARTO PROPORCIONAL

```
def reparto_proporcional(M,valores):
    S = sum(valores)
    return [M*(v/S) for v in valores]


M = 1000
```

```
valores = [2,3,5]
print(reparto_proporcional(M,valores))
```

---

3. REPARTO CON NOMBRES

- Asociar **nombres** a los valores para identificar el reparto:

```
M = 1000
personas = ["Ana","Luis","Marta"]
valores = [2,3,5]

S = sum(valores)
for nombre, v in zip(personas,valores):
    parte = M*(v/S)
    print(f"{nombre} recibe: {parte:.2f}")
```

Resultado:
Ana recibe: 200.00
Luis recibe: 300.00
Marta recibe: 500.00

---

4. REPARTO REDONDEADO

- Para manejar **monedas o cantidades enteras**, se puede redondear:

```
M = 1000
valores = [2,3,5]

S = sum(valores)
partes = [round(M*(v/S),2) for v in valores]
print(partes)
```

Resultado: [200.0, 300.0, 500.0]

---

5. RESUMEN

- **Parte proporcional = M × (vi / suma_total)**
- Se puede usar **listas, bucles y zip** para manejar nombres y valores
- Redondear si se requiere **cantidad entera o moneda**
- Python permite automatizar **repartos complejos** de forma clara

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE REPARTO PROPORCIONAL

1. Reparto M=1000, valores=[2,3,5]

```
M=1000; v=[2,3,5]; S=sum(v); print([M*(x/S) for x in v])
```

2. Reparto con nombres

```
M=1000; personas=["Ana","Luis","Marta"]; v=[2,3,5]; S=sum(v)
for n,x in zip(personas,v): print(f"{n}: {M*(x/S):.2f}")
```

3. Reparto M=500, valores=[1,2,2]

```
M=500; v=[1,2,2]; S=sum(v); print([M*(x/S) for x in v])
```

4. Reparto M=1200, valores=[3,5,2]

```
M=1200; v=[3,5,2]; S=sum(v); print([M*(x/S) for x in v])
```

5. Función reparto proporcional

```
def reparto(M,v): S=sum(v); return [M*(x/S) for x in v]
print(reparto(1000,[2,3,5]))
```

6. Reparto con redondeo

```
M=1000; v=[2,3,5]; S=sum(v); print([round(M*(x/S),2) for x in v])
```

7. Reparto M=750, valores=[4,1,5]

```
M=750; v=[4,1,5]; S=sum(v); print([M*(x/S) for x in v])
```

8. Reparto con nombres y redondeo

```
M=1000; personas=["A","B","C"]; v=[2,3,5]; S=sum(v)
for n,x in zip(personas,v): print(f"{n}: {round(M*(x/S),2)}")
```

9. Reparto M=600, valores=[1,1,1]

```
M=600; v=[1,1,1]; S=sum(v); print([M*(x/S) for x in v])
```

10. Reparto M=850, valores=[3,2]

```
M=850; v=[3,2]; S=sum(v); print([M*(x/S) for x in v])
```

11. Reparto M=900, valores=[1,2,3,4]

```
M=900; v=[1,2,3,4]; S=sum(v); print([M*(x/S) for x in v])
```

12. Función con nombres

```
def reparto_nombres(M,nom,v):
    S=sum(v)
    for n,x in zip(nom,v): print(f"{n}: {M*(x/S):.2f}")
reparto_nombres(1000,["Ana","Luis","Marta"],[2,3,5])
```

13. Reparto M=1000, valores=[10,20,30]

```
M=1000; v=[10,20,30]; S=sum(v); print([M*(x/S) for x in v])
```

14.Redondeo M=1000, valores=[10,20,30]

```
M=1000; v=[10,20,30]; S=sum(v); print([round(M*(x/S),2) for x in v])
```

15.Reparto M=500, valores=[5,5,5,5]

```
M=500; v=[5,5,5,5]; S=sum(v); print([M*(x/S) for x in v])
```

16.Reparto con nombres M=1200, valores=[4,6,10]

```
M=1200; nom=["A","B","C"]; v=[4,6,10]; S=sum(v)
for n,x in zip(nom,v): print(f"{n}: {M*(x/S):.2f}")
```

17.Reparto M=1000, valores=[7,3]

```
M=1000; v=[7,3]; S=sum(v); print([M*(x/S) for x in v])
```

18.Reparto M=1000, valores=[1,2,3]

```
M=1000; v=[1,2,3]; S=sum(v); print([M*(x/S) for x in v])
```

19.Redondeo reparto M=1000, valores=[1,2,3]

```
M=1000; v=[1,2,3]; S=sum(v); print([round(M*(x/S),2) for x in v])
```

20.Función completa con redondeo

```
def reparto_final(M,v): S=sum(v); return [round(M*(x/S),2) for x in v]
print(reparto_final(1000,[2,3,5]))
```

---

30. LECCIÓN: REGLA DE COMPAÑÍA

INTRODUCCIÓN

En el contexto de una **compañía**, Python permite manejar información sobre **empleados, salarios, ventas, inventario y reparto de utilidades**.
Con Python se puede:

- Crear **listas y diccionarios** para almacenar datos de empleados o productos
- Calcular **salarios, bonificaciones y comisiones**
- Automatizar **reportes financieros y operativos**
- Aplicar **funciones y bucles** para procesar información de la compañía

---

1. ALMACENAMIENTO DE DATOS DE EMPLEADOS

Se puede usar un **diccionario** para almacenar información de cada empleado:

```
empleado = {
    "nombre": "Ana",
    "cargo": "Gerente",
    "salario": 2000,
    "ventas": 5000
}
print(empleado)
```

- **Clave:** atributo (nombre, cargo, salario)
- **Valor:** dato correspondiente

---

2. LISTA DE EMPLEADOS

```
empleados = [
    {"nombre":"Ana","cargo":"Gerente","salario":2000,"ventas":5000},
    {"nombre":"Luis","cargo":"Vendedor","salario":1500,"ventas":3000},
    {"nombre":"Marta","cargo":"Vendedor","salario":1500,"ventas":4000}
]


for e in empleados:
    print(f"{e['nombre']} - {e['cargo']} - {e['salario']} - {e['ventas']}")
```

---

3. CALCULAR BONIFICACIÓN POR VENTAS

```
for e in empleados:
```

```
    bonificacion = e['ventas']*0.05  # 5% sobre ventas
    total = e['salario'] + bonificacion
    print(f"{e['nombre']} recibe un total de: {total}")
```

---

4. FUNCIONES PARA GESTIÓN DE EMPLEADOS

```
def total_empleado(empleado, porcentaje_venta):
    return empleado['salario'] + empleado['ventas']*porcentaje_venta

for e in empleados:
    print(f"{e['nombre']}: {total_empleado(e,0.05)}")
```

---

5. SUMA DE SALARIOS DE LA COMPAÑÍA

```
total_salarios = sum(e['salario'] for e in empleados)
print(f"Total salarios: {total_salarios}")
```

---

6. REPARTO DE UTILIDADES

Si la compañía decide repartir utilidades proporcional a ventas:

```
utilidades = 3000
total_ventas = sum(e['ventas'] for e in empleados)

for e in empleados:
    parte = utilidades * (e['ventas']/total_ventas)
    print(f"{e['nombre']} recibe: {parte:.2f} de utilidades")
```

---

7. CREAR FUNCIONES PARA REPORTE DE COMPAÑÍA

```
def reporte_utilidades(empleados, utilidades):
    total_ventas = sum(e['ventas'] for e in empleados)
    for e in empleados:
        parte = utilidades*(e['ventas']/total_ventas)
        print(f"{e['nombre']} recibe {parte:.2f} de utilidades")

reporte_utilidades(empleados,3000)
```

---

8. RESUMEN

- **Diccionarios y listas** permiten almacenar información de empleados
- Se pueden calcular **salarios totales, bonificaciones y utilidades**

- **Bucles y funciones** automatizan procesos
- Python es útil para **reportes financieros y gestión de compañías**

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE COMPAÑÍA

1. Crear diccionario de empleado

```
empleado={"nombre":"Ana","cargo":"Gerente","salario":2000,"ventas":5000}
print(empleado)
```

2. Lista de empleados

```
empleados=[{"nombre":"Ana","cargo":"Gerente","salario":2000,"ventas":5000},
{"nombre":"Luis","cargo":"Vendedor","salario":1500,"ventas":3000}]
print(empleados)
```

3. Imprimir nombres de empleados

```
for e in empleados: print(e['nombre'])
```

4. Calcular total con bonificación 5%

```
for e in empleados: print(e['nombre'], e['salario']+e['ventas']*0.05)
```

5. Función total empleado

```
def total_empleado(emp): return emp['salario']+emp['ventas']*0.05
for e in empleados: print(e['nombre'], total_empleado(e))
```

6. Total salarios

```
print(sum(e['salario'] for e in empleados))
```

7. Total ventas

```
print(sum(e['ventas'] for e in empleados))
```

8. Reparto utilidades 3000

```
util=3000; total_ventas=sum(e['ventas'] for e in empleados)
for e in empleados: print(e['nombre'], util*(e['ventas']/total_ventas))
```

9. Crear función reparto utilidades

```
def reparto_util(emp,util): total=sum(e['ventas'] for e in emp)
for e in emp: print(e['nombre'], util*(e['ventas']/total))
reparto_util(empleados,3000)
```

10. Añadir nuevo empleado

```
empleados.append({"nombre":"Marta","cargo":"Vendedor","salario":1500,"ventas":40
00})
print(empleados)
```

11. Calcular bonificación total

```
total_bon=sum(e['ventas']*0.05 for e in empleados)
```

```
print(total_bon)
```

12.Empleados con ventas > 3500

```
for e in empleados:
    if e['ventas']>3500: print(e['nombre'])
```

13.Salario más alto

```
max_sal=max(e['salario'] for e in empleados)
print(max_sal)
```

14.Empleado con salario más alto

```
for e in empleados:
    if e['salario']==max_sal: print(e['nombre'])
```

15.Ordenar empleados por ventas

```
empleados.sort(key=lambda x:x['ventas'], reverse=True)
for e in empleados: print(e['nombre'], e['ventas'])
```

16.Sumar salarios + bonificaciones

```
total=sum(e['salario']+e['ventas']*0.05 for e in empleados)
print(total)
```

17.Crear reporte de salarios totales

```
for e in empleados: print(f"{e['nombre']} total={e['salario']
+e['ventas']*0.05}")
```

18.Reducir ventas de un empleado

```
empleados[1]['ventas']-=500
print(empleados[1])
```

19.Calcular porcentaje de ventas

```
total_ventas=sum(e['ventas'] for e in empleados)
for e in empleados: print(e['nombre'], e['ventas']/total_ventas)
```

20.Función completa de reporte

```
def reporte_completo(emp):
    total_ventas=sum(e['ventas'] for e in emp)
    for e in emp: print(f"{e['nombre']} salario={e['salario']}
ventas={e['ventas']} utilidades={3000*(e['ventas']/total_ventas):.2f}")
reporte_completo(empleados)
```

---

31. LECCIÓN: ALIGACIÓN O MEZCLA

INTRODUCCIÓN

La **aligación o mezcla** es un concepto utilizado en matemáticas y química para **combinar dos o más sustancias con diferentes concentraciones** y obtener una concentración final deseada.
Python permite:

- Calcular la **cantidad de cada componente**
- Determinar la **concentración final**
- Automatizar cálculos con **listas, bucles y funciones**
- Aplicar fórmulas a **problemas de mezcla de soluciones, alimentos o productos químicos**

---

1. FÓRMULA GENERAL DE ALIGACIÓN

Para mezclar dos sustancias:

C1 × V1 + C2 × V2 = Cf × Vf

Donde:

- **C1, C2** = concentraciones de los componentes
- **V1, V2** = volúmenes de cada componente
- **Cf** = concentración final deseada
- **Vf = V1 + V2** = volumen total

Se puede despejar:

- Para obtener **V1**:
  V1 = (Cf - C2) / (C1 - C2) × Vf
- Para obtener **V2**:
  V2 = Vf - V1

---

2. EJEMPLO BÁSICO

```
C1 = 40  # Concentración componente 1
C2 = 20  # Concentración componente 2
Cf = 30  # Concentración deseada
Vf = 100 # Volumen total

V1 = (Cf - C2)/(C1 - C2) * Vf
V2 = Vf - V1

print("Volumen componente 1:", V1)
```

```
print("Volumen componente 2:", V2)
```

Resultado:
Volumen componente 1: 50.0
Volumen componente 2: 50.0

---

3. FUNCIONES PARA ALIGACIÓN

```
def aligacion(C1,C2,Cf,Vf):
    V1 = (Cf - C2)/(C1 - C2) * Vf
    V2 = Vf - V1
    return V1, V2


V1, V2 = aligacion(40,20,30,100)
print("V1:", V1, "V2:", V2)
```

---

4. REPARTO PROPORCIONAL (REGLA DE LA ALIGACIÓN)

- La **regla de la aligación** indica que los volúmenes son proporcionales a las **diferencias de concentración**:

```
C1, C2, Cf = 40, 20, 30
Vf = 100


V1 = (Cf - C2) * Vf / (C1 - C2)
V2 = Vf - V1
print(V1, V2)
```

- Esto es útil para **mezclas de dos o más componentes**

---

5. ALIGACIÓN CON MÁS DE DOS COMPONENTES

- Para **varios componentes**, se puede usar Python con **listas y bucles**:

```
concentraciones = [50,30,20]
volumen_total = 100
concentracion_final = 35


volumenes = [(concentracion_final - C)/(concentraciones[0] - C) * volumen_total
for C in concentraciones[1:]]
print(volumenes)
```

- Esto calcula los volúmenes de cada componente en base a la concentración deseada.

---

6. RESUMEN

- **Aligación = combinar componentes para obtener concentración deseada**
- Fórmula básica: **C1*V1* + *C2*V2 = Cf*Vf**
- Python permite automatizar con **funciones, bucles y listas**
- Se pueden resolver **problemas de mezcla de soluciones, alimentos o químicos**

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE ALIGACIÓN

1. Mezcla C1=40%, C2=20%, Cf=30%, Vf=100

```
C1,C2,Cf,Vf=40,20,30,100; V1=(Cf-C2)/(C1-C2)*Vf; V2=Vf-V1; print(V1,V2)
```

2. Mezcla C1=60%, C2=10%, Cf=40%, Vf=200

```
C1,C2,Cf,Vf=60,10,40,200; V1=(Cf-C2)/(C1-C2)*Vf; V2=Vf-V1; print(V1,V2)
```

3. Mezcla C1=50%, C2=30%, Cf=40%, Vf=150

```
C1,C2,Cf,Vf=50,30,40,150; V1=(Cf-C2)/(C1-C2)*Vf; V2=Vf-V1; print(V1,V2)
```

4. Mezcla C1=80%, C2=20%, Cf=50%, Vf=100

```
C1,C2,Cf,Vf=80,20,50,100; V1=(Cf-C2)/(C1-C2)*Vf; V2=Vf-V1; print(V1,V2)
```

5. Mezcla C1=70%, C2=30%, Cf=55%, Vf=200

```
C1,C2,Cf,Vf=70,30,55,200; V1=(Cf-C2)/(C1-C2)*Vf; V2=Vf-V1; print(V1,V2)
```

6. Mezcla C1=90%, C2=10%, Cf=60%, Vf=300

```
C1,C2,Cf,Vf=90,10,60,300; V1=(Cf-C2)/(C1-C2)*Vf; V2=Vf-V1; print(V1,V2)
```

7. Mezcla C1=45%, C2=15%, Cf=30%, Vf=120

```
C1,C2,Cf,Vf=45,15,30,120; V1=(Cf-C2)/(C1-C2)*Vf; V2=Vf-V1; print(V1,V2)
```

8. Mezcla C1=55%, C2=25%, Cf=40%, Vf=80

```
C1,C2,Cf,Vf=55,25,40,80; V1=(Cf-C2)/(C1-C2)*Vf; V2=Vf-V1; print(V1,V2)
```

9. Mezcla C1=100%, C2=0%, Cf=25%, Vf=50

```
C1,C2,Cf,Vf=100,0,25,50; V1=(Cf-C2)/(C1-C2)*Vf; V2=Vf-V1; print(V1,V2)
```

10. Mezcla C1=35%, C2=15%, Cf=25%, Vf=60

```
C1,C2,Cf,Vf=35,15,25,60; V1=(Cf-C2)/(C1-C2)*Vf; V2=Vf-V1; print(V1,V2)
```

11. Función aligación dos componentes

```
def alig(C1,C2,Cf,Vf): V1=(Cf-C2)/(C1-C2)*Vf; return V1,Vf-V1
print(alig(40,20,30,100))
```

12. Mezcla C1=75%, C2=25%, Cf=50%, Vf=200

```
C1,C2,Cf,Vf=75,25,50,200; V1=(Cf-C2)/(C1-C2)*Vf; V2=Vf-V1; print(V1,V2)
```

13. Mezcla C1=60%, C2=20%, Cf=40%, Vf=150

```
C1,C2,Cf,Vf=60,20,40,150; V1=(Cf-C2)/(C1-C2)*Vf; V2=Vf-V1; print(V1,V2)
```

14.Mezcla C1=50%, C2=0%, Cf=25%, Vf=100

```
C1,C2,Cf,Vf=50,0,25,100; V1=(Cf-C2)/(C1-C2)*Vf; V2=Vf-V1; print(V1,V2)
```

15.Mezcla C1=80%, C2=40%, Cf=60%, Vf=120

```
C1,C2,Cf,Vf=80,40,60,120; V1=(Cf-C2)/(C1-C2)*Vf; V2=Vf-V1; print(V1,V2)
```

16.Mezcla C1=90%, C2=10%, Cf=50%, Vf=200

```
C1,C2,Cf,Vf=90,10,50,200; V1=(Cf-C2)/(C1-C2)*Vf; V2=Vf-V1; print(V1,V2)
```

17.Mezcla C1=60%, C2=30%, Cf=45%, Vf=180

```
C1,C2,Cf,Vf=60,30,45,180; V1=(Cf-C2)/(C1-C2)*Vf; V2=Vf-V1; print(V1,V2)
```

18.Mezcla C1=50%, C2=20%, Cf=35%, Vf=140

```
C1,C2,Cf,Vf=50,20,35,140; V1=(Cf-C2)/(C1-C2)*Vf; V2=Vf-V1; print(V1,V2)
```

19.Mezcla C1=70%, C2=30%, Cf=50%, Vf=100

```
C1,C2,Cf,Vf=70,30,50,100; V1=(Cf-C2)/(C1-C2)*Vf; V2=Vf-V1; print(V1,V2)
```

20.Función aligación con impresión detallada

```
def alig_det(C1,C2,Cf,Vf): V1=(Cf-C2)/(C1-C2)*Vf; V2=Vf-V1; print(f"V1={V1}, V2={V2}"); return V1,V2

alig_det(40,20,30,100)
```

---

32. LECCIÓN: ALEACIONES

INTRODUCCIÓN

Una **aleación** es una mezcla de **dos o más metales u otros elementos** que se combinan para obtener ciertas propiedades físicas o químicas.
Python permite:

- Calcular **porcentajes de cada componente**
- Determinar la **masa de cada metal en la aleación**
- Automatizar cálculos con **listas, bucles y funciones**
- Aplicar cálculos a problemas de **metales preciosos, industria y química**

---

1. FÓRMULA GENERAL DE ALEACIÓN

Si tenemos una aleación con **M** gramos y queremos obtener una concentración **C%** de un metal:

Masa del metal = M × (C / 100)

Para mezclar dos aleaciones y obtener una aleación final:

M1 × C1 + M2 × C2 = Mf × Cf

Donde:

- **M1, M2** = masa de cada componente
- **C1, C2** = concentración del metal en cada componente
- **Mf = M1 + M2** = masa total
- **Cf** = concentración final deseada

Se puede despejar para obtener:

M1 = (Cf - C2) / (C1 - C2) × Mf
M2 = Mf - M1

---

2. EJEMPLO BÁSICO DE ALEACIÓN

```
C1 = 70  # Concentración aleación 1 (%)
C2 = 40  # Concentración aleación 2 (%)
Cf = 60  # Concentración deseada (%)
Mf = 100 # Masa total


M1 = (Cf - C2)/(C1 - C2) * Mf
M2 = Mf - M1
```

```
print("Masa aleación 1:", M1)
print("Masa aleación 2:", M2)
```

Resultado:
Masa aleación 1: 66.67
Masa aleación 2: 33.33

---

3. FUNCIONES PARA ALEACIONES

```
def aleacion(C1,C2,Cf,Mf):
    M1 = (Cf - C2)/(C1 - C2) * Mf
    M2 = Mf - M1
    return M1, M2


M1, M2 = aleacion(70,40,60,100)
print("M1:", M1, "M2:", M2)
```

---

4. REGLA DE MEZCLA PROPORCIONAL

- Los volúmenes o masas se calculan **proporcionales a la diferencia de concentración**:

```
C1, C2, Cf = 70, 40, 60
Mf = 100


M1 = (Cf - C2) * Mf / (C1 - C2)
M2 = Mf - M1
print(M1, M2)
```

- Útil para **mezclar varias aleaciones y obtener concentración deseada**

---

5. ALEACIÓN CON VARIOS COMPONENTES

- Para **más de dos componentes**, se puede usar **listas y bucles** en Python:

```
concentraciones = [80,50,30]
masa_total = 150
concentracion_final = 60


masas = [(concentracion_final - C)/(concentraciones[0] - C) * masa_total for C
in concentraciones[1:]]
print(masas)
```

---

6. RESUMEN

- **Aleación = mezcla de metales con concentración deseada**
- Fórmula básica: **M1*C1* + *M2*C2 = Mf*Cf**
- Python permite automatizar cálculos con **funciones, listas y bucles**
- Se pueden resolver **problemas de aleaciones metálicas y químicas**

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE ALEACIONES

1. Aleación C1=70%, C2=40%, Cf=60%, Mf=100

```
C1,C2,Cf,Mf=70,40,60,100; M1=(Cf-C2)/(C1-C2)*Mf; M2=Mf-M1; print(M1,M2)
```

2. Aleación C1=80%, C2=50%, Cf=65%, Mf=200

```
C1,C2,Cf,Mf=80,50,65,200; M1=(Cf-C2)/(C1-C2)*Mf; M2=Mf-M1; print(M1,M2)
```

3. Aleación C1=90%, C2=60%, Cf=75%, Mf=150

```
C1,C2,Cf,Mf=90,60,75,150; M1=(Cf-C2)/(C1-C2)*Mf; M2=Mf-M1; print(M1,M2)
```

4. Aleación C1=70%, C2=50%, Cf=60%, Mf=120

```
C1,C2,Cf,Mf=70,50,60,120; M1=(Cf-C2)/(C1-C2)*Mf; M2=Mf-M1; print(M1,M2)
```

5. Aleación C1=60%, C2=30%, Cf=45%, Mf=100

```
C1,C2,Cf,Mf=60,30,45,100; M1=(Cf-C2)/(C1-C2)*Mf; M2=Mf-M1; print(M1,M2)
```

6. Aleación C1=50%, C2=20%, Cf=35%, Mf=200

```
C1,C2,Cf,Mf=50,20,35,200; M1=(Cf-C2)/(C1-C2)*Mf; M2=Mf-M1; print(M1,M2)
```

7. Aleación C1=80%, C2=40%, Cf=60%, Mf=300

```
C1,C2,Cf,Mf=80,40,60,300; M1=(Cf-C2)/(C1-C2)*Mf; M2=Mf-M1; print(M1,M2)
```

8. Aleación C1=90%, C2=50%, Cf=70%, Mf=250

```
C1,C2,Cf,Mf=90,50,70,250; M1=(Cf-C2)/(C1-C2)*Mf; M2=Mf-M1; print(M1,M2)
```

9. Aleación C1=75%, C2=25%, Cf=50%, Mf=180

```
C1,C2,Cf,Mf=75,25,50,180; M1=(Cf-C2)/(C1-C2)*Mf; M2=Mf-M1; print(M1,M2)
```

10. Aleación C1=65%, C2=35%, Cf=50%, Mf=200

```
C1,C2,Cf,Mf=65,35,50,200; M1=(Cf-C2)/(C1-C2)*Mf; M2=Mf-M1; print(M1,M2)
```

11. Función aleación dos componentes

```
def alea(C1,C2,Cf,Mf): M1=(Cf-C2)/(C1-C2)*Mf; return M1,Mf-M1
print(alea(70,40,60,100))
```

12. Aleación C1=85%, C2=55%, Cf=70%, Mf=120

```
C1,C2,Cf,Mf=85,55,70,120; M1=(Cf-C2)/(C1-C2)*Mf; M2=Mf-M1; print(M1,M2)
```

13. Aleación C1=60%, C2=30%, Cf=45%, Mf=150

```
C1,C2,Cf,Mf=60,30,45,150; M1=(Cf-C2)/(C1-C2)*Mf; M2=Mf-M1; print(M1,M2)
```

14.Aleación C1=70%, C2=30%, Cf=50%, Mf=200

```
C1,C2,Cf,Mf=70,30,50,200; M1=(Cf-C2)/(C1-C2)*Mf; M2=Mf-M1; print(M1,M2)
```

15.Aleación C1=55%, C2=25%, Cf=40%, Mf=180

```
C1,C2,Cf,Mf=55,25,40,180; M1=(Cf-C2)/(C1-C2)*Mf; M2=Mf-M1; print(M1,M2)
```

16.Aleación C1=95%, C2=50%, Cf=75%, Mf=300

```
C1,C2,Cf,Mf=95,50,75,300; M1=(Cf-C2)/(C1-C2)*Mf; M2=Mf-M1; print(M1,M2)
```

17.Aleación C1=80%, C2=60%, Cf=70%, Mf=200

```
C1,C2,Cf,Mf=80,60,70,200; M1=(Cf-C2)/(C1-C2)*Mf; M2=Mf-M1; print(M1,M2)
```

18.Aleación C1=75%, C2=50%, Cf=65%, Mf=150

```
C1,C2,Cf,Mf=75,50,65,150; M1=(Cf-C2)/(C1-C2)*Mf; M2=Mf-M1; print(M1,M2)
```

19.Aleación C1=60%, C2=20%, Cf=40%, Mf=100

```
C1,C2,Cf,Mf=60,20,40,100; M1=(Cf-C2)/(C1-C2)*Mf; M2=Mf-M1; print(M1,M2)
```

20.Función aleación con impresión detallada

```
def alea_det(C1,C2,Cf,Mf): M1=(Cf-C2)/(C1-C2)*Mf; M2=Mf-M1; print(f"M1={M1}, M2={M2}"); return M1,M2

alea_det(70,40,60,100)
```

---

33. LECCIÓN: PROMEDIOS

INTRODUCCIÓN

El **promedio** es un valor que representa la **media aritmética** de un conjunto de números.
En Python, se puede calcular fácilmente usando **listas, bucles y funciones**.
Se utiliza en:

- **Notas y calificaciones**
- **Estadísticas de ventas o producción**
- **Resultados de encuestas o mediciones**
- **Cualquier conjunto de datos numéricos**

---

1. PROMEDIO ARITMÉTICO

La fórmula:

Promedio = (Suma de todos los valores) / (Cantidad de valores)

Ejemplo básico:

```
numeros = [10, 20, 30, 40, 50]
promedio = sum(numeros)/len(numeros)
print("Promedio:", promedio)
```

Resultado: 30.0

---

2. FUNCIONES PARA PROMEDIO

```
def calcular_promedio(lista):
    return sum(lista)/len(lista)

numeros = [15, 25, 35, 45]
print("Promedio:", calcular_promedio(numeros))
```

---

3. PROMEDIO PONDERADO

Si cada valor tiene un **peso diferente**:

Promedio ponderado = (x1*w1* + *x2*w2 + ... + xn*wn) / (w1 + w2 + ... + wn)

Ejemplo:

```
valores = [80, 90, 70]
pesos = [2, 3, 1]
prom_pond = sum(v*p for v,p in zip(valores,pesos))/sum(pesos)
print("Promedio ponderado:", prom_pond)
```

---

4. PROMEDIO CON DATOS INGRESADOS POR USUARIO

```
numeros = []
for i in range(5):
    num = float(input("Ingrese un número: "))
    numeros.append(num)

promedio = sum(numeros)/len(numeros)
print("Promedio:", promedio)
```

---

5. PROMEDIO DE DICCIONARIOS

Para datos como calificaciones de alumnos:

```
notas = {"Ana":80, "Luis":90, "Marta":70}
promedio = sum(notas.values())/len(notas)
print("Promedio:", promedio)
```

---

6. RESUMEN

- **Promedio aritmético:** suma de valores dividido entre cantidad
- **Promedio ponderado:** toma en cuenta los pesos de cada valor
- Python permite usar **listas, diccionarios, bucles y funciones**
- Es útil para **análisis de datos, notas, ventas o mediciones**

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE PROMEDIOS

1. Promedio de lista simple

```
numeros=[10,20,30,40,50]; print(sum(numeros)/len(numeros))
```

2. Promedio de lista distinta

```
numeros=[15,25,35,45,55]; print(sum(numeros)/len(numeros))
```

3. Promedio de números negativos

```
numeros=[-10,20,-30,40]; print(sum(numeros)/len(numeros))
```

4. Promedio de un solo número

```
numeros=[100]; print(sum(numeros)/len(numeros))
```

5. Promedio de decimales

```
numeros=[1.5,2.5,3.5,4.5]; print(sum(numeros)/len(numeros))
```

6. Promedio ponderado

```
valores=[80,90,70]; pesos=[2,3,1]; print(sum(v*p for v,p in
zip(valores,pesos))/sum(pesos))
```

7. Promedio ponderado distinto

```
valores=[50,60,70]; pesos=[1,2,3]; print(sum(v*p for v,p in
zip(valores,pesos))/sum(pesos))
```

8. Promedio usando función

```
def prom(lista): return sum(lista)/len(lista); print(prom([10,20,30]))
```

9. Promedio de diccionario

```
notas={"Ana":80,"Luis":90,"Marta":70}; print(sum(notas.values())/len(notas))
```

10. Promedio con bucle for

```
numeros=[5,15,25]; total=0

for n in numeros: total+=n

print(total/len(numeros))
```

11. Promedio de 10 números

```
numeros=list(range(1,11)); print(sum(numeros)/len(numeros))
```

12. Promedio de números ingresados

```
numeros=[float(input()) for _ in range(3)]; print(sum(numeros)/len(numeros))
```

13. Promedio de lista con negativos y positivos

```
numeros=[-5,10,-15,20]; print(sum(numeros)/len(numeros))
```

14. Promedio de decimales con función

```
def prom_dec(lista): return sum(lista)/len(lista)

print(prom_dec([1.2,3.4,5.6]))
```

15. Promedio de lista larga

```
numeros=list(range(50,101,10)); print(sum(numeros)/len(numeros))
```

16. Promedio de lista aleatoria

```
import random; numeros=[random.randint(1,100) for _ in range(5)];
print(sum(numeros)/len(numeros))
```

17. Promedio ponderado de notas de examen

```
notas=[85,90,75]; pesos=[3,4,2]; print(sum(v*p for v,p in
zip(notas,pesos))/sum(pesos))
```

18. Promedio con redondeo

```
numeros=[10,20,30]; print(round(sum(numeros)/len(numeros),2))
```

19. Promedio de varias listas

```
listas=[[10,20],[30,40],[50,60]]; total=sum(sum(l) for l in listas);
cantidad=sum(len(l) for l in listas); print(total/cantidad)
```

20. Promedio en diccionario con pesos

```
notas={"Ana":80,"Luis":90,"Marta":70}; pesos={"Ana":2,"Luis":3,"Marta":1};
print(sum(notas[n]*pesos[n] for n in notas)/sum(pesos.values()))
```

---

34. LECCIÓN: MONEDA

INTRODUCCIÓN

El manejo de **moneda** en Python se refiere a trabajar con **cantidades de dinero, conversiones, cálculos de cambio, sumas y descuentos**.
Python permite:

- Representar cantidades monetarias usando **floats** o librerías especializadas como `decimal`
- Realizar **operaciones aritméticas** precisas
- Calcular **conversiones de moneda**
- Automatizar **reportes financieros o ventas**

---

1. REPRESENTAR MONEDA EN PYTHON

```
precio = 150.75
print("Precio:", precio)
```

- Para evitar errores de redondeo, se recomienda usar la librería **decimal**:

```
from decimal import Decimal


precio = Decimal('150.75')
print("Precio:", precio)
```

---

2. SUMA Y RESTA DE MONEDA

```
compra1 = Decimal('150.50')
compra2 = Decimal('200.25')
total = compra1 + compra2
print("Total:", total)
```

- **Resta:** `cambio = pago - total`

---

3. MULTIPLICACIÓN Y DIVISIÓN

- Multiplicar por cantidad de artículos:

```
precio_unitario = Decimal('20.50')
cantidad = 3
total = precio_unitario * cantidad
print("Total:", total)
```

- Dividir, por ejemplo, para pagar entre varias personas:

```
total = Decimal('100.50')
```

```
personas = 3
por_persona = total / personas
print("Por persona:", por_persona)
```

---

4. REDONDEO DE MONEDA

```
total = Decimal('100.567')
total_redondeado = round(total, 2)
print("Total redondeado:", total_redondeado)
```

- Normalmente se redondea a **2 decimales**

---

5. CONVERSIÓN DE MONEDA

```
usd = Decimal('100')
tasa_euro = Decimal('0.92')
euro = usd * tasa_euro
print("Equivalente en euros:", euro)
```

- Se puede hacer con listas para **varias monedas**

---

6. LISTAS Y DICCIONARIOS DE MONEDA

```
precios = [Decimal('10.50'), Decimal('20.25'), Decimal('30.75')]
total = sum(precios)
print("Total:", total)

cambio = {"USD": 100, "EUR": 92, "GBP": 80}
for moneda, valor in cambio.items():
    print(moneda, valor)
```

---

7. FUNCIONES PARA OPERACIONES DE MONEDA

```
def calcular_total(precios):
    return sum(precios)

precios = [Decimal('15.50'), Decimal('25.75'), Decimal('10.25')]
print("Total:", calcular_total(precios))
```

---

8. RESUMEN

- **Decimal** es recomendado para manejar dinero con precisión
- Se pueden hacer **suma, resta, multiplicación, división y redondeo**

- Python permite **conversiones de monedas, cálculos de cambio y reportes financieros**
- **Funciones, listas y diccionarios** facilitan la organización de la información

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE MONEDA

1. Sumar dos cantidades

```
from decimal import Decimal
total = Decimal('150.50') + Decimal('200.25'); print(total)
```

2. Restar dinero

```
from decimal import Decimal
cambio = Decimal('500') - Decimal('123.45'); print(cambio)
```

3. Multiplicar precio por cantidad

```
from decimal import Decimal
total = Decimal('20.50') * 3; print(total)
```

4. Dividir entre personas

```
from decimal import Decimal
por_persona = Decimal('100.50') / 3; print(por_persona)
```

5. Redondeo a 2 decimales

```
from decimal import Decimal
total = Decimal('100.567'); print(round(total,2))
```

6. Conversión USD a EUR

```
from decimal import Decimal
usd=Decimal('100'); euro=usd*Decimal('0.92'); print(euro)
```

7. Lista de precios y suma

```
from decimal import Decimal
precios=[Decimal('10.50'),Decimal('20.25'),Decimal('30.75')];
print(sum(precios))
```

8. Diccionario de monedas

```
cambio={"USD":100,"EUR":92,"GBP":80}; print(cambio["EUR"])
```

9. Total con función

```
from decimal import Decimal
def total(lista): return sum(lista)
print(total([Decimal('15.50'),Decimal('25.75')]))
```

10. Pago y cambio

```
from decimal import Decimal
pago=Decimal('500'); total=Decimal('123.45'); print(pago-total)
```

11.Precio con descuento 10%

```
from decimal import Decimal
precio=Decimal('200'); descuento=precio*Decimal('0.10'); print(precio-descuento)
```

12.IVA 21%

```
from decimal import Decimal
precio=Decimal('100'); iva=precio*Decimal('0.21'); print(precio+iva)
```

13.Convertir lista a dólares

```
from decimal import Decimal
tasas=[Decimal('0.92'),Decimal('0.80')]; cantidad=Decimal('100')
for t in tasas: print(cantidad*t)
```

14.Precio por persona con redondeo

```
from decimal import Decimal
total=Decimal('100.567'); personas=3; print(round(total/personas,2))
```

15.Suma de pagos

```
from decimal import Decimal
pagos=[Decimal('50.25'),Decimal('75.50'),Decimal('100.75')]; print(sum(pagos))
```

16.Conversión GBP a USD

```
from decimal import Decimal
gbp=Decimal('80'); usd=gbp*Decimal('1.25'); print(usd)
```

17.Multiplicación de lista

```
from decimal import Decimal
precios=[Decimal('10.50'),Decimal('20.25')]; cantidades=[2,3]
total=sum(p*q for p,q in zip(precios,cantidades)); print(total)
```

18.Redondeo de cada precio en lista

```
from decimal import Decimal
precios=[Decimal('10.567'),Decimal('20.234')]; print([round(p,2) for p in
precios])
```

19.Promedio de gastos

```
from decimal import Decimal
gastos=[Decimal('50'),Decimal('100'),Decimal('150')];
print(sum(gastos)/len(gastos))
```

20.Total de ventas con impuestos

```
from decimal import Decimal
ventas=[Decimal('100'),Decimal('200'),Decimal('300')]; total=sum(ventas);
total_impuestos=total*Decimal('1.21'); print(total_impuestos)
```

---

30 monedas importantes del mundo y su tipo de cambio frente al euro (1 EUR = … moneda), basada en los datos de referencia del European Central Bank (ECB). (European Central Bank)

Moneda	Código	Tipo de cambio (aprox.)
Dólar estadounidense	USD	1 EUR ≈ 1.1741 USD (European Central Bank)
Yen japonés	JPY	1 EUR ≈ 173.76 JPY (European Central Bank)
Lev búlgaro	BGN	1 EUR ≈ 1.9558 BGN (European Central Bank)
Corona checa	CZK	1 EUR ≈ 24.335 CZK (European Central Bank)
Corona danesa	DKK	1 EUR ≈ 7.4649 DKK (European Central Bank)
Libra esterlina británica	GBP	1 EUR ≈ 0.87340 GBP (European Central Bank)
Forinto húngaro	HUF	1 EUR ≈ 390.26 HUF (European Central Bank)
Zloty polaco	PLN	1 EUR ≈ 4.2698 PLN (European Central Bank)
Leu rumano	RON	1 EUR ≈ 5.0806 RON (European Central Bank)
Corona sueca	SEK	1 EUR ≈ 11.0565 SEK (European Central Bank)
Franco suizo	CHF	1 EUR ≈ 0.9364 CHF (European Central Bank)
Corona islandesa	ISK	1 EUR ≈ 142.40 ISK (European Central Bank)
Corona noruega	NOK	1 EUR ≈ 11.7265 NOK (European Central Bank)
Lira turca	TRY	1 EUR ≈ 48.8227 TRY (European Central Bank)
Dólar australiano	AUD	1 EUR ≈ 1.7760 AUD (European Central Bank)
Real brasileño	BRL	1 EUR ≈ 6.2432 BRL (European Central Bank)
Dólar canadiense	CAD	1 EUR ≈ 1.6346 CAD (European Central Bank)
Yuan chino (renminbi)	CNY	1 EUR ≈ 8.3591 CNY (European Central Bank)
Dólar de Hong Kong	HKD	1 EUR ≈ 9.1359 HKD (European Central Bank)
Rupia indonesia	IDR	1 EUR ≈ 19,578.94 IDR (European Central Bank)
Shekel israelí	ILS	1 EUR ≈ 3.8786 ILS (European Central Bank)
Rupia india	INR	1 EUR ≈ 104.2548 INR (European Central Bank)
Won surcoreano	KRW	1 EUR ≈ 1,648.05 KRW (European Central Bank)
Peso mexicano	MXN	1 EUR ≈ 21.5314 MXN (European Central Bank)
Ringgit malayo	MYR	1 EUR ≈ 4.9412 MYR (European Central Bank)
Dólar neozelandés	NZD	1 EUR ≈ 2.0257 NZD (European Central Bank)
Peso filipino	PHP	1 EUR ≈ 68.375 PHP (European Central Bank)
Dólar de Singapur	SGD	1 EUR ≈ 1.5145 SGD (European Central Bank)
Baht tailandés	THB	1 EUR ≈ 38.088 THB (European Central Bank)
Rand sudafricano	ZAR	1 EUR ≈ 20.2820 ZAR (European Central Bank)

35. LECCIÓN: REGLA CONJUNTA

INTRODUCCIÓN

La **regla conjunta** es un método matemático utilizado para resolver problemas de **proporciones múltiples** o combinadas, donde intervienen varias cantidades relacionadas entre sí.
En Python, podemos resolver problemas de regla conjunta usando:

- **Variables**
- **Operadores aritméticos** (+, -, *, /)
- **Funciones**
- **Listas y diccionarios** para almacenar datos

Se aplica en problemas como:

- Distribución de cantidades según proporciones
- Mezclas de productos
- Comparaciones de precios, tiempos o cantidades en conjunto

---

1. PROBLEMAS BÁSICOS DE REGLA CONJUNTA

Ejemplo:
Si 3 máquinas producen 180 piezas en 4 horas, ¿cuántas piezas producirán 5 máquinas en 6 horas?

```
# Datos
maquinas1 = 3
piezas1 = 180
horas1 = 4

maquinas2 = 5
horas2 = 6

# Regla conjunta: piezas2 = piezas1 * (maquinas2/maquinas1) * (horas2/horas1)
piezas2 = piezas1 * (maquinas2/maquinas1) * (horas2/horas1)
print("Piezas producidas:", piezas2)
```

Resultado: 450 piezas

---

2. FUNCIÓN GENERAL PARA REGLA CONJUNTA

```
def regla_conjunta(cantidad1, factor1, factor2, factor3):
    return cantidad1 * (factor1/factor2) * factor3
```

```
# Ejemplo
piezas = regla_conjunta(180, 5, 3, 6)
print("Piezas producidas:", piezas)
```

---

3. PROBLEMAS CON MEZCLAS DE CANTIDADES

Ejemplo:
Si 2 litros de solución A se mezclan con 3 litros de solución B, ¿cuánto de A y B necesito para obtener 10 litros de mezcla?

```
A = 2
B = 3
total_deseado = 10
factor = total_deseado / (A + B)
A_nuevo = A * factor
B_nuevo = B * factor
print("Cantidad A:", A_nuevo)
print("Cantidad B:", B_nuevo)
```

---

4. PROBLEMAS CON TIEMPO Y TRABAJO

Si varias personas trabajan juntas, la regla conjunta permite calcular cuánto tiempo tardarán:

```
personas1 = 4
tiempo1 = 6
personas2 = 6

tiempo2 = (personas1 * tiempo1) / personas2
print("Tiempo requerido:", tiempo2)
```

---

5. RESUMEN DE PASOS

1. Identificar todas las **cantidades relacionadas**
2. Determinar cuáles **aumentan o disminuyen** proporcionalmente
3. Aplicar la **fórmula general**:

Resultado = Cantidad inicial × (Factor proporcional 1) × (Factor proporcional 2) ...

4. Para mezclas, calcular **factor de proporción total**
5. Usar **funciones y variables** para automatizar cálculos

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE REGLA CONJUNTA

1. Producción de máquinas

```
piezas=180*5/3*6; print(piezas)
```

2. Mezcla de líquidos

```
A,B=2,3; total=10; f=total/(A+B); print(A*f,B*f)
```

3. Tiempo de trabajo de personas

```
print(4*6/6)
```

4. Distribución proporcional de dinero

```
total=120; a,b=3,2; f=total/(a+b); print(a*f,b*f)
```

5. Producción por horas y máquinas

```
print(200*5/4*8/5)
```

6. Mezcla de ingredientes

```
azucar,harina=3,7; total=50; f=total/(azucar+harina); print(azucar*f,harina*f)
```

7. Rendimiento de trabajadores

```
print(5*8/4*6/3)
```

8. Reparto de trabajo según tiempo

```
t1=6;t2=10;trabajo=300;personas1=5;personas2=7;print(trabajo*personas2/
personas1*t1/t2)
```

9. Cantidad de pintura

```
rojo,azul=2,5;total=28;f=total/(rojo+azul);print(rojo*f,azul*f)
```

10. Producción conjunta de fábricas

```
fabrica1,fabrica2=3,4;horas1,horas2=8,10;piezas1=240;print(piezas1*fabrica2/
fabrica1*horas2/horas1)
```

11. Mezcla de jugos

```
naranja,manzana=3,2;total=25;f=total/
(naranja+manzana);print(naranja*f,manzana*f)
```

12. Reparto proporcional de premios

```
print(500*3/5,500*2/5)
```

13. Trabajo en equipo

```
personas1,personas2=4,6;tiempo1,tiempo2=5,0;print(personas1*tiempo1/personas2)
```

14. Mezcla de soluciones químicas

```
A,B=7,3;total=50;f=total/(A+B);print(A*f,B*f)
```

15. Producción combinada de máquinas

```
print(120*5/4*3/2)
```

16. Tiempo requerido por más personas

```
print(8*5/10)
```

17.Mezcla de café

```
arabica,robusta=4,6;total=20;f=total/
(arabica+robusta);print(arabica*f,robusta*f)
```

18.Reparto de materiales

```
ladrillos=500;hombres1,hombres2=5,10;dias1,dias2=8,0;print(ladrillos*hombres2/
hombres1*dias2/dias1)
```

19.Mezcla de leche y agua

```
leche,agua=3,2;total=25;f=total/(leche+agua);print(leche*f,agua*f)
```

20.Producción múltiple de fábricas y horas

```
piezas=150*3/2*6/4; print(piezas)
```

---

36. LECCIÓN: SEGUROS

INTRODUCCIÓN

Los **seguros** implican el cálculo de **primas, coberturas, porcentajes de riesgo, indemnizaciones** y pueden involucrar conceptos de **porcentaje, proporcionalidad y tiempo**.
En Python, podemos resolver problemas de seguros usando:

- **Variables**
- **Operadores aritméticos** (+, -, *, /)
- **Funciones**
- **Listas y diccionarios** para almacenar datos

Los problemas comunes incluyen:

- Cálculo de la **prima de seguro**
- Distribución de riesgos
- Cálculo de indemnizaciones por siniestro
- Ajustes de primas según porcentajes o periodos

---

1. CÁLCULO DE PRIMA BÁSICA

```
# Datos
valor_asegurado = 10000
tasa_prima = 0.05  # 5%

# Prima = valor asegurado * tasa
prima = valor_asegurado * tasa_prima
print("Prima a pagar:", prima)
```

---

2. PRIMA ANUALIZADA

```
# Prima mensual a anual
prima_mensual = 50
prima_anual = prima_mensual * 12
print("Prima anual:", prima_anual)
```

---

3. INDEMNIZACIÓN POR SINIESTRO

```
valor_asegurado = 15000
porcentaje_indemnizacion = 0.8  # 80% del valor
indemnizacion = valor_asegurado * porcentaje_indemnizacion
```

```
print("Indemnización:", indemnizacion)
```

---

4. SEGURO CON FRANQUICIA

```
valor_siniestro = 5000
franquicia = 1000
pago_aseguradora = max(valor_siniestro - franquicia, 0)
print("Pago de la aseguradora:", pago_aseguradora)
```

---

5. DISTRIBUCIÓN DE PRIMAS EN GRUPO

```
total_prima = 1200
personas = 4
prima_por_persona = total_prima / personas
print("Prima por persona:", prima_por_persona)
```

---

6. FUNCIÓN PARA CALCULAR PRIMA

```
def calcular_prima(valor, tasa):
    return valor * tasa

print(calcular_prima(10000, 0.06))
```

---

7. SEGUROS CON AÑOS DE VIGENCIA

```
valor_asegurado = 20000
tasa_anual = 0.04
años = 3
prima_total = valor_asegurado * tasa_anual * años
print("Prima total:", prima_total)
```

---

8. RESUMEN DE PASOS

1. Identificar **valor asegurado** y **tasa de prima**
2. Aplicar **operaciones aritméticas** para obtener prima o indemnización
3. Considerar **franquicias, periodos y porcentajes de cobertura**
4. Usar **funciones y listas** para cálculos repetitivos o grupos

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE SEGUROS

1. Prima básica

```
print(10000*0.05)
```

2. Prima anualizada de mensual

```
print(50*12)
```

3. Indemnización 80%

```
print(15000*0.8)
```

4. Pago con franquicia

```
print(max(5000-1000,0))
```

5. Distribución de prima en 4 personas

```
print(1200/4)
```

6. Prima con función

```
def prima(valor,tasa): return valor*tasa
print(prima(10000,0.06))
```

7. Prima total por 3 años

```
print(20000*0.04*3)
```

8. Prima con descuento 10%

```
prima=1000; print(prima*0.9)
```

9. Indemnización 50%

```
valor=8000; print(valor*0.5)
```

10. Pago con franquicia mayor al siniestro

```
print(max(500-1000,0))
```

11. Suma de primas de 3 pólizas

```
print(500+750+1000)
```

12. Prima proporcional a meses

```
valor=12000;tasa=0.06;meses=6;print(valor*tasa*meses/12)
```

13. Cálculo de riesgo por porcentaje

```
riesgo=0.02;valor=15000;print(valor*riesgo)
```

14. Prima por grupo de 5 personas

```
total=1500;print(total/5)
```

15. Indemnización parcial 30%

```
print(20000*0.3)
```

16. Prima con tasa variable

```
print(10000*0.05*1.2)
```

17. Pago fraccionado en 3 cuotas

```
total=1200;print(total/3)
```

18.Indemnización con deducible

```
valor=7000;deducible=500;print(max(valor-deducible,0))
```

19.Prima acumulada 2 años

```
print(10000*0.05*2)
```

20.Total primas de varias pólizas con tasas distintas

```
print(10000*0.05 + 15000*0.04 + 20000*0.03)
```

---

37. LECCIÓN: ESTADÍSTICA

INTRODUCCIÓN

La **estadística** es la rama de las matemáticas que estudia la **recolección, análisis, interpretación y presentación de datos**.
En Python, podemos resolver problemas estadísticos usando:

- **Listas** para almacenar datos
- **Operadores aritméticos**
- **Funciones**
- Bibliotecas como **statistics** y **numpy** para cálculos avanzados

Algunos conceptos básicos:

- **Media (promedio)**
- **Mediana**
- **Moda**
- **Rango, varianza y desviación estándar**

---

1. LISTAS DE DATOS

```
datos = [5, 7, 3, 8, 10, 5]
print("Datos:", datos)
```

---

2. MEDIA (PROMEDIO)

```
datos = [5, 7, 3, 8, 10, 5]
media = sum(datos) / len(datos)
print("Media:", media)
```

---

3. MEDIANA

```
datos = [5, 7, 3, 8, 10, 5]
datos.sort()
n = len(datos)
if n % 2 == 0:
    mediana = (datos[n//2 - 1] + datos[n//2]) / 2
else:
    mediana = datos[n//2]
print("Mediana:", mediana)
```

---

4. MODA

```
from collections import Counter
datos = [5, 7, 3, 8, 10, 5]
frecuencia = Counter(datos)
moda = frecuencia.most_common(1)[0][0]
print("Moda:", moda)
```

---

5. RANGO

```
datos = [5, 7, 3, 8, 10, 5]
rango = max(datos) - min(datos)
print("Rango:", rango)
```

---

6. VARIANZA Y DESVIACIÓN ESTÁNDAR

```
import statistics
datos = [5, 7, 3, 8, 10, 5]
varianza = statistics.variance(datos)
desviacion = statistics.stdev(datos)
print("Varianza:", varianza)
print("Desviación estándar:", desviacion)
```

---

7. FRECUENCIA RELATIVA

```
datos = [5, 7, 3, 8, 10, 5]
frecuencia = {x: datos.count(x)/len(datos) for x in set(datos)}
print("Frecuencia relativa:", frecuencia)
```

---

8. REPRESENTACIÓN CON LISTAS Y DICCIONARIOS

```
datos = [5,7,3,8,10,5]
estadistica = {
    "media": sum(datos)/len(datos),
    "mediana": sorted(datos)[len(datos)//2],
    "moda": max(set(datos), key=datos.count),
    "rango": max(datos)-min(datos)
}
print(estadistica)
```

RESUMEN DE PASOS

1. **Recolectar datos** en listas o diccionarios
2. Calcular **media, mediana y moda**
3. Calcular **rango, varianza y desviación estándar**
4. Analizar la **frecuencia y proporciones**
5. Usar **funciones y bibliotecas** para simplificar cálculos

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE ESTADÍSTICA

1. Media simple

```
datos=[5,7,3,8,10,5]; print(sum(datos)/len(datos))
```

2. Mediana

```
datos=[5,7,3,8,10,5]; datos.sort(); n=len(datos); print((datos[n//2-
1]+datos[n//2])/2)
```

3. Moda

```
from collections import Counter; datos=[5,7,3,8,10,5];
print(Counter(datos).most_common(1)[0][0])
```

4. Rango

```
datos=[5,7,3,8,10,5]; print(max(datos)-min(datos))
```

5. Varianza

```
import statistics; datos=[5,7,3,8,10,5]; print(statistics.variance(datos))
```

6. Desviación estándar

```
import statistics; datos=[5,7,3,8,10,5]; print(statistics.stdev(datos))
```

7. Frecuencia relativa

```
datos=[5,7,3,8,10,5]; print({x:datos.count(x)/len(datos) for x in set(datos)})
```

8. Suma de datos

```
datos=[5,7,3,8,10,5]; print(sum(datos))
```

9. Número de datos

```
datos=[5,7,3,8,10,5]; print(len(datos))
```

10. Media ponderada

```
valores=[5,7,3]; pesos=[2,1,3]; print(sum(v*p for v,p in
zip(valores,pesos))/sum(pesos))
```

11. Diferencia con la media

```
datos=[5,7,3]; media=sum(datos)/len(datos); print([x-media for x in datos])
```

12. Productos de diferencias

```
datos=[5,7,3]; media=sum(datos)/len(datos); print([(x-media)**2 for x in datos])
```

13.Frecuencia absoluta

```
datos=[5,7,3,5]; print({x:datos.count(x) for x in set(datos)})
```

14.Media de notas

```
notas=[8,7,9,10]; print(sum(notas)/len(notas))
```

15.Mediana con número impar

```
datos=[5,7,3]; datos.sort(); n=len(datos); print(datos[n//2])
```

16.Desviación manual

```
datos=[5,7,3]; media=sum(datos)/len(datos); print((sum((x-media)**2 for x in
datos)/(len(datos)-1))**0.5)
```

17.Rango con min/max

```
datos=[2,4,6,8]; print(max(datos)-min(datos))
```

18.Moda múltiple

```
datos=[5,5,7,7,3]; from collections import Counter; c=Counter(datos);
maxf=max(c.values()); print([k for k,v in c.items() if v==maxf])
```

19.Frecuencia acumulada

```
datos=[3,5,7]; datos.sort(); print([sum(1 for x in datos if x<=i) for i in
datos])
```

20.Resumen estadístico

```
datos=[5,7,3,8,10,5]; import statistics;
print({"media":sum(datos)/len(datos),"mediana":statistics.median(datos),"moda":s
tatistics.mode(datos),"rango":max(datos)-
min(datos),"varianza":statistics.variance(datos),"desviacion":statistics.stdev(d
atos)})
```

---

38. LECCIÓN: OPERACIONES BINARIAS

INTRODUCCIÓN

Las **operaciones binarias** son aquellas que se realizan sobre **números binarios** (base 2).
En Python podemos manejar números binarios usando:

- **Prefijo `0b`** para declarar números binarios
- **Operadores aritméticos** (+, -, *, /)
- **Operadores bit a bit** (&, |, ^, ~, <<, >>)
- Funciones de conversión: **`bin()`**, **`int()`**

Las operaciones más importantes son:

- **Suma y resta binaria**
- **Multiplicación y división binaria**
- **AND, OR, XOR, NOT**
- **Desplazamiento de bits**

---

1. REPRESENTACIÓN BINARIA EN PYTHON

```
a = 0b1010  # 10 en decimal
b = 0b1101  # 13 en decimal
print("a:", a, "b:", b)
```

---

2. CONVERSIÓN ENTRE DECIMAL Y BINARIO

```
decimal = 25
binario = bin(decimal)
print("Decimal:", decimal, "Binario:", binario)

binario = 0b11001
decimal = int(binario)
print("Binario:", binario, "Decimal:", decimal)
```

---

3. SUMA Y RESTA BINARIA

```
a = 0b1010
b = 0b1101
suma = a + b
resta = b - a
print("Suma:", bin(suma))
```

```
print("Resta:", bin(resta))
```

---

4. MULTIPLICACIÓN Y DIVISIÓN BINARIA

```
a = 0b1010
b = 0b11
multiplicacion = a * b
division = a // b
print("Multiplicación:", bin(multiplicacion))
print("División entera:", bin(division))
```

---

5. OPERACIONES BIT A BIT

- **AND (&)**: devuelve 1 si ambos bits son 1
- **OR (|)**: devuelve 1 si al menos un bit es 1
- **XOR (^)**: devuelve 1 si los bits son diferentes
- **NOT (~)**: invierte los bits

```
a = 0b1010
b = 0b1101
print("AND:", bin(a & b))
print("OR:", bin(a | b))
print("XOR:", bin(a ^ b))
print("NOT a:", bin(~a))
```

---

6. DESPLAZAMIENTO DE BITS

- **<<**: desplaza a la izquierda (multiplica por 2^n)
- **>>**: desplaza a la derecha (divide por 2^n)

```
a = 0b1010
print("Izquierda 2:", bin(a << 2))
print("Derecha 2:", bin(a >> 2))
```

---

7. RESUMEN DE PASOS

1. Representar números en binario usando `0b`
2. Convertir entre decimal y binario usando `bin()` o `int()`
3. Aplicar operaciones aritméticas normalmente
4. Aplicar operaciones bit a bit para manipulación de bits

5. Usar desplazamientos para multiplicaciones o divisiones rápidas por potencias de 2

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE OPERACIONES BINARIAS

1. Suma binaria

```
print(bin(0b1010 + 0b1101))
```

2. Resta binaria

```
print(bin(0b1101 - 0b1010))
```

3. Multiplicación binaria

```
print(bin(0b101 * 0b11))
```

4. División entera binaria

```
print(bin(0b1100 // 0b10))
```

5. AND binario

```
print(bin(0b1010 & 0b1101))
```

6. OR binario

```
print(bin(0b1010 | 0b1101))
```

7. XOR binario

```
print(bin(0b1010 ^ 0b1101))
```

8. NOT binario

```
print(bin(~0b1010))
```

9. Desplazamiento a la izquierda

```
print(bin(0b1010 << 3))
```

10. Desplazamiento a la derecha

```
print(bin(0b1010 >> 2))
```

11. Conversión decimal a binario

```
print(bin(45))
```

12. Conversión binario a decimal

```
print(int(0b101101))
```

13. Suma de varios binarios

```
print(bin(0b1010 + 0b110 + 0b111))
```

14. AND múltiple

```
print(bin(0b1010 & 0b1111 & 0b1101))
```

15. OR múltiple

```
print(bin(0b1010 | 0b0011 | 0b0101))
```

16.XOR múltiple

```
print(bin(0b1010 ^ 0b1100 ^ 0b0110))
```

17.Desplazamiento combinado

```
a = 0b1011
print(bin((a << 2) >> 1))
```

18.Complemento de un binario

```
a = 0b1101
print(bin(~a))
```

19.Operación mixta

```
a,b = 0b1010,0b0110
print(bin((a & b) | (a ^ b)))
```

20.Verificación de bit específico

```
a = 0b101010
bit = (a >> 3) & 1
print("Bit 3:", bit)
```

---

39. LECCIÓN: COORDENADAS

INTRODUCCIÓN

En **Python**, podemos trabajar con **coordenadas** para representar **puntos en el plano cartesiano** o en **espacio tridimensional**.
Las coordenadas se usan en:

- Gráficos y visualización de datos
- Geometría computacional
- Juegos y simulaciones
- Sistemas de mapas y GPS

Conceptos básicos:

- **Coordenadas 2D:** (x, y)
- **Coordenadas 3D:** (x, y, z)
- **Distancia entre puntos**
- **Traslación, rotación y escalamiento**

---

1. REPRESENTACIÓN DE COORDENADAS

```
# Coordenada 2D
punto2D = (3, 5)
print("Punto 2D:", punto2D)

# Coordenada 3D
punto3D = (2, 4, 6)
print("Punto 3D:", punto3D)
```

---

2. ACCESO A LOS VALORES DE LAS COORDENADAS

```
x, y = punto2D
print("x:", x, "y:", y)

x, y, z = punto3D
print("x:", x, "y:", y, "z:", z)
```

---

3. DISTANCIA ENTRE DOS PUNTOS 2D

```
import math
p1 = (1, 2)
p2 = (4, 6)
```

```
distancia = math.sqrt((p2[0]-p1[0])**2 + (p2[1]-p1[1])**2)
print("Distancia entre p1 y p2:", distancia)
```

---

4. DISTANCIA ENTRE DOS PUNTOS 3D

```
p1 = (1, 2, 3)
p2 = (4, 6, 8)
distancia = math.sqrt((p2[0]-p1[0])**2 + (p2[1]-p1[1])**2 + (p2[2]-p1[2])**2)
print("Distancia 3D:", distancia)
```

---

5. TRASLACIÓN DE UN PUNTO

```
punto = (3, 5)
traslacion = (2, -1)
nuevo_punto = (punto[0]+traslacion[0], punto[1]+traslacion[1])
print("Punto trasladado:", nuevo_punto)
```

---

6. ESCALAMIENTO DE UN PUNTO

```
punto = (3, 5)
factor = 2
nuevo_punto = (punto[0]*factor, punto[1]*factor)
print("Punto escalado:", nuevo_punto)
```

---

7. ROTACIÓN DE UN PUNTO EN 2D

```
punto = (3, 4)
angulo = math.radians(90)  # convertir grados a radianes
x_rot = punto[0]*math.cos(angulo) - punto[1]*math.sin(angulo)
y_rot = punto[0]*math.sin(angulo) + punto[1]*math.cos(angulo)
print("Punto rotado 90°:", (x_rot, y_rot))
```

---

8. LISTAS DE PUNTOS Y OPERACIONES

```
puntos = [(1,2),(3,4),(5,6)]
# Trasladar todos los puntos
traslacion = (1,1)
puntos_trasladados = [(x+traslacion[0], y+traslacion[1]) for x,y in puntos]
print("Puntos trasladados:", puntos_trasladados)
```

RESUMEN DE PASOS

1. Representar **coordenadas** con **tuplas o listas**
2. Acceder a los valores con **desempaquetado**
3. Calcular **distancias** usando la **fórmula de Euclides**
4. Aplicar **traslaciones, rotaciones y escalamiento** usando operaciones aritméticas y trigonometría
5. Usar **listas** y comprensión de listas para manipular múltiples puntos

---

20 PROGRAMAS NUMERADOS Y RESUELTOS DE COORDENADAS

1. Crear punto 2D

```
p=(3,5); print(p)
```

2. Crear punto 3D

```
p=(2,4,6); print(p)
```

3. Acceder a coordenadas 2D

```
p=(3,5); x,y=p; print(x,y)
```

4. Acceder a coordenadas 3D

```
p=(2,4,6); x,y,z=p; print(x,y,z)
```

5. Distancia 2D

```
import math; p1=(1,2); p2=(4,6); print(math.sqrt((p2[0]-p1[0])**2+(p2[1]-p1[1])**2))
```

6. Distancia 3D

```
import math; p1=(1,2,3); p2=(4,6,8); print(math.sqrt((p2[0]-p1[0])**2+(p2[1]-p1[1])**2+(p2[2]-p1[2])**2))
```

7. Traslación de punto

```
p=(3,5); t=(2,-1); print((p[0]+t[0], p[1]+t[1]))
```

8. Escalamiento de punto

```
p=(3,5); f=2; print((p[0]*f,p[1]*f))
```

9. Rotación 90°

```
import math; p=(3,4); a=math.radians(90); print((p[0]*math.cos(a)-p[1]*math.sin(a), p[0]*math.sin(a)+p[1]*math.cos(a)))
```

10. Lista de puntos

```
p=[(1,2),(3,4),(5,6)]; print(p)
```

11. Traslación de lista de puntos

```
p=[(1,2),(3,4)]; t=(1,1); print([(x+t[0],y+t[1]) for x,y in p])
```

12. Escalamiento de lista de puntos

```
p=[(1,2),(3,4)]; f=2; print([(x*f,y*f) for x,y in p])
```

13.Suma de coordenadas

```
p1=(1,2); p2=(3,4); print((p1[0]+p2[0], p1[1]+p2[1]))
```

14.Resta de coordenadas

```
p1=(5,7); p2=(2,3); print((p1[0]-p2[0],p1[1]-p2[1]))
```

15.Distancia horizontal

```
p1=(1,2); p2=(4,2); print(abs(p2[0]-p1[0]))
```

16.Distancia vertical

```
p1=(1,2); p2=(1,5); print(abs(p2[1]-p1[1]))
```

17.Punto medio

```
p1=(1,2); p2=(5,6); print(((p1[0]+p2[0])/2,(p1[1]+p2[1])/2))
```

18.Rotación 180°

```
import math; p=(3,4); a=math.radians(180); print((p[0]*math.cos(a)-
p[1]*math.sin(a),p[0]*math.sin(a)+p[1]*math.cos(a)))
```

19.Traslación negativa

```
p=(3,5); t=(-2,-3); print((p[0]+t[0],p[1]+t[1]))
```

20.Escalamiento negativo

```
p=(2,4); f=-1; print((p[0]*f,p[1]*f))
```

Epílogo

Al llegar al final de este libro, no solo concluyes una serie de capítulos: culminas un recorrido que une dos disciplinas fundamentales del conocimiento humano—las matemáticas y la programación—en un mismo camino de exploración y creación. Si has seguido cada concepto, cada ejercicio y cada fragmento de código, ahora dispones de una herramienta poderosa: la capacidad de transformar ideas abstractas en soluciones concretas utilizando Python.

Las matemáticas te han mostrado su rostro más práctico, revelándose no como fórmulas aisladas, sino como una estructura lógica que da forma a algoritmos, modelos y programas. Python, por su parte, ha sido el puente que te permitió experimentar con esos conceptos, manipularlos, representarlos y descubrir que la programación no es más que otra forma de pensar matemáticamente.

Pero este final no pretende cerrar puertas, sino abrirlas. Lo aprendido aquí es solo el inicio. Hay mundos enteros esperando: cálculo avanzado, álgebra lineal aplicada, estadística computacional, inteligencia artificial, optimización… En todos ellos, las matemáticas y Python seguirán siendo tus aliados inseparables.

Ojalá este libro te haya inspirado a seguir aprendiendo, creando y preguntando. Que cada programa que escribas sea una ventana nueva y que cada idea matemática que descubras se convierta en una invitación a profundizar. La verdadera magia ocurre cuando combinamos pensamiento lógico, creatividad y curiosidad; y tú ya tienes las bases para seguir construyendo.

Gracias por haber recorrido este camino. Que tu pasión por las matemáticas y Python continúe creciendo y te acompañe en todos tus futuros proyectos.